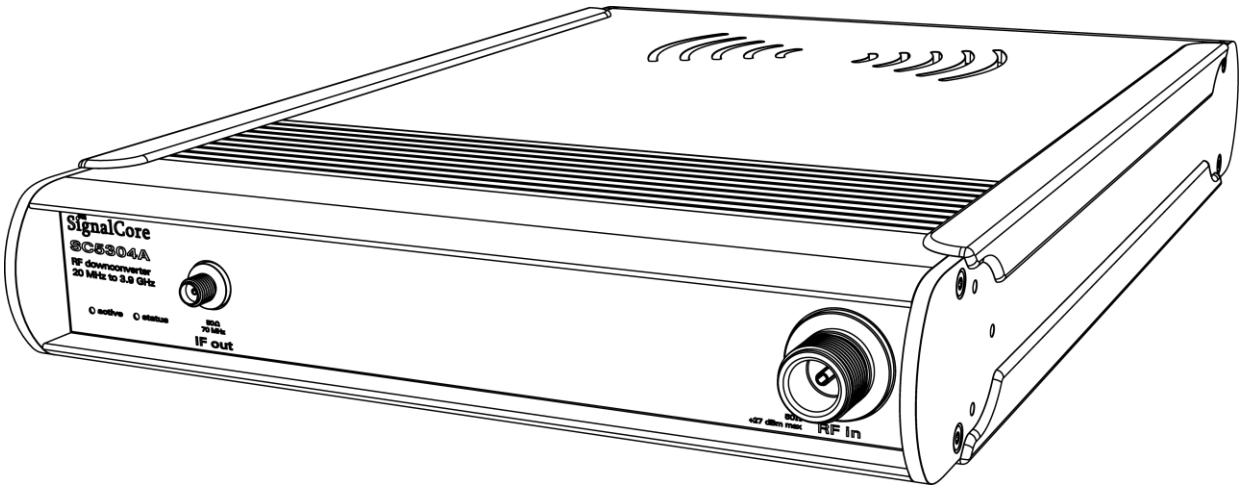
SignalCore™
PRESERVING SIGNAL INTEGRITY



SC5304A
1 MHz to 3.9 GHz RF Downconverter with USB Interface

Operating & Programming Manual

© 2012 SignalCore, Inc.
support@signalcore.com

CONTENTS

Important Information

Warranty 1

Copyright & Trademarks 2

International Materials Declarations 2

CE European Union EMC & Safety Compliance Declaration 2

Recycling Information 3

Warnings Regarding Use of SignalCore Products 3

Getting Started

Unpacking 4

Verifying the Contents of your Shipment 4

Setting Up and Configuring the SC5304A 4

Signal Connections 6

Indicator LEDs 6

USB Communication Connection 6

Power Switch 7

Fuse 7

Power Connection 7

SC5304A-Theory of Operation

Overview 8

Signal Path Description 9

Local Oscillator Description 12

Frequency Tuning Modes 13

Setting the SC5304A to Achieve Best Dynamic Range 14

Operating the SC5304A Outside Normal Range 15

SC5304A Programming Interface

Device Drivers 16

The SC5304A USB Configuration 16

Writing the Device Registers 17

Reading the Device Registers 17

Using the Application Programming Interface (API) 17

Setting the SC5304A: Writing Configuration Registers Directly

Configuration Registers 18
Tuning the RF Frequency 18
Changing the Attenuator Settings 19
MSB value 19
Attenuator 19
Enabling and Disabling the RF Preamplifier 19
Changing the RF Synthesizer Mode 19
Selecting the IF Filter Path 19
Setting the Reference Clock Behavior 20
Adjusting the Reference Clock Accuracy 20
Setting Spectral Inversion in the IF 20
Storing Data into the User EEPROM Space 20
Setting the Phase of the IF Signal 20

Querying the SC5304A: Writing Request Registers Directly

Reading Device Status Data 21
Reading Temperature Data 22
Reading Calibration EEPROM Data 23
Reading User EEPROM Data 24

Working With Calibration Data

EEPROM Data Content 26
Frequency Correction 28
Gain Correction 28
IF Response Correction 30

Software API Library Functions

Constants Definitions 33
Type Definitions 34
Function Definitions and Usage 35

Calibration & Maintenance 50

SC5304A Accessories

Port Accessories 51

Ordering Information 51

Appendix A - Specifications **52**

IMPORTANT INFORMATION

Warranty

The warranty terms and conditions for all SignalCore products are also provided on our corporate website. Please visit http://www.signalcore.com/ for more information.

This product is warranted against defects in materials and workmanship for a period of one year from the date of shipment. SignalCore will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

Before any equipment will be accepted for warranty repair or replacement, a Return Material Authorization (RMA) number must be obtained from a SignalCore customer service representative and clearly marked on the outside of the return package. SignalCore will pay all shipping costs relating to warranty repair or replacement.

SignalCore strives to make the information in this document as accurate as possible. The document has been carefully reviewed for technical and typographic accuracy. In the event that technical or typographical errors exist, SignalCore reserves the right to make changes to subsequent editions of this document without prior notice to possessors of this edition. Please contact SignalCore if errors are suspected. In no event shall SignalCore be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SIGNALCORE, INCORPORATED MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SIGNALCORE, INCORPORATED SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SIGNALCORE, INCORPORATED WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of SignalCore, Incorporated will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against SignalCore, Incorporated must be brought within one year after the cause of action accrues. SignalCore, Incorporated shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow SignalCore, Incorporated's installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright & Trademarks

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of SignalCore, Incorporated.

SignalCore, Incorporated respects the intellectual property rights of others, and we ask those who use our products to do the same. Our products are protected by copyright and other intellectual property laws. Use of SignalCore products is restricted to applications that do not infringe on the intellectual property rights of others.

"SignalCore", "signalcore.com", and the phrase "preserving signal integrity" are registered trademarks of SignalCore, Incorporated. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

International Materials Declarations

SignalCore, Incorporated uses a fully RoHS compliant manufacturing process for our products. Therefore, SignalCore hereby declares that its products do not contain restricted materials as defined by European Union directive 2002/95/EC (EU RoHS) in any amounts higher than limits stated in the directive. This statement is based on the assumption of reliable information and data provided by our component suppliers and may not have been independently verified through other means. For products sold into China, we also comply with the "Administrative Measure on the Control of Pollution Caused by Electronic Information Products" (China RoHS). In the current stage of this legislation, the content of six hazardous materials must be explicitly declared. Each of those materials, and the categorical amount present in our products, are shown below:

組成名稱 **Model Name**	鉛 **Lead (Pb)**	汞 **Mercury (Hg)**	镉 **Cadmium (Cd)**	六价铬 **Hexavalent Chromium (Cr(VI))**	多溴联苯 **Polybrominated biphenyls (PBB)**	多溴二苯醚 **Polybrominated diphenyl ethers (PBDE)**
SC5304A	✓	✓	✓	✓	✓	✓

A ✓ indicates that the hazardous substance contained in all of the homogeneous materials for this product is below the limit requirement in SJ/T11363-2006. An **X** indicates that the particular hazardous substance contained in at least one of the homogeneous materials used for this product is above the limit requirement in SJ/T11363-2006.

CE European Union EMC & Safety Compliance Declaration

The European Conformity (CE) marking is affixed to products with input of 50 - 1,000 VAC or 75 - 1,500 VDC and/or for products which may cause or be affected by electromagnetic disturbance. The CE marking symbolizes conformity of the product with the applicable requirements. CE compliance is a manufacturer's self-declaration allowing products to circulate freely within the European Union (EU). SignalCore products meet the essential requirements of Directives 2004/108/EC (EMC) and 2006/95/EC

(product safety), and comply with the relevant standards. Standards for Measurement, Control and Laboratory Equipment include EN 61326 and EN 55011 for EMC, and EN 61010-1 for product safety.

Recycling Information

All products sold by SignalCore eventually reach the end of their useful life. SignalCore complies with EU directive 2002/96/EC regarding Waste Electrical and Electronic Equipment (WEEE).

Warnings Regarding Use of SignalCore Products

(1) PRODUCTS FOR SALE BY SIGNALCORE, INCORPORATED ARE NOT DESIGNED WITH COMPONENTS NOR TESTED FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED “SYSTEM FAILURES”). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE SOLELY RELIANT UPON ANY ONE COMPONENT DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM SIGNALCORE' TESTING PLATFORMS, AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE SIGNALCORE PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY SIGNALCORE, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF SIGNALCORE PRODUCTS WHENEVER SIGNALCORE PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

GETTING STARTED

Unpacking

All SignalCore products ship in antistatic packaging (bags) to prevent damage from electrostatic discharge (ESD). Under certain conditions, an ESD event can instantly and permanently damage several of the components found in SignalCore products. Therefore, to avoid damage when handling any SignalCore hardware, you must take the following precautions:

- Ground yourself using a grounding strap or by touching a grounded metal object.
- Touch the antistatic bag to a grounded metal object before removing the hardware from its packaging.
- *Never* touch exposed signal pins. Due to the inherent performance degradation caused by ESD protection circuits in the RF path, the device has minimal ESD protection against direct injection of ESD into the RF signal pins.
- When not in use, store all SignalCore products in their original antistatic bags.

Remove the product from its packaging and inspect it for loose components or any signs of damage. Notify SignalCore immediately if the product appears damaged in any way.

Verifying the Contents of your Shipment

In addition to this User Guide, verify that your SC5304A kit contains the following items:

Quantity	Item
1	SC5304A USB RF Downconverter
1	IEC Power Cord
1	USB cable
1	Software Installation CD

Setting Up and Configuring the SC5304A

The SC5304A is a rugged, actively-cooled device for either benchtop or rack mounted use. However, even an actively-cooled device still requires attention to ensure the device can properly shed the heat it generates. Inadequate airflow can cause the temperature inside the RF housing to rise above the maximum for this product, leading to improper performance and potentially reducing product lifespan or causing complete product failure. Maintain adequate air space around the unit at all times.

Cooling for this device is sufficient when the on-board temperature sensor indicates a rise of no more than 20 °C above ambient temperature under normal operating conditions.

The SC5304A is a benchtop subsystem downconverter with user I/O located on both the front and rear faces of the unit as shown in Figure 1 and Figure 2. Each location is discussed in further detail below.

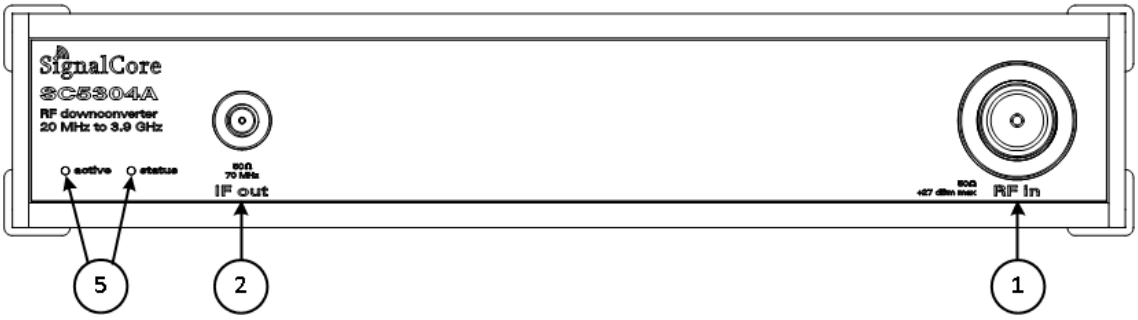


Figure 1. Front view of the SC5304A.

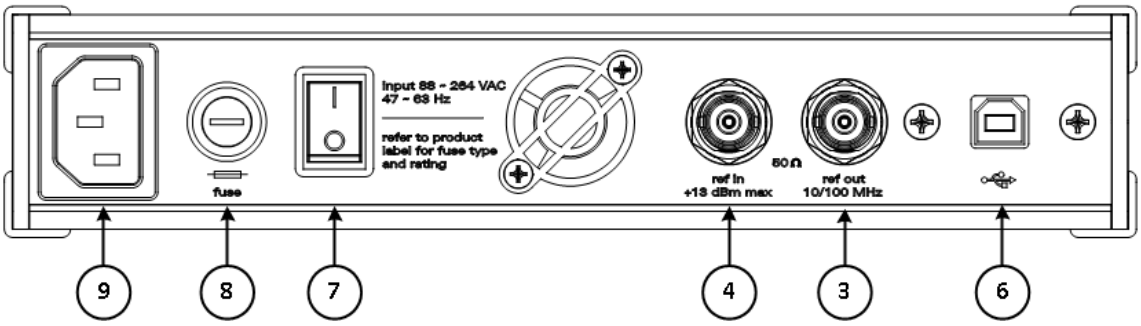


Figure 2. Rear view of the SC5304A.

Signal connections (ports) to the SC5304A are of three types: N, SMA, and BNC. Exercise caution when fastening cables to the signal connections. Over-tightening any connection can cause permanent damage to the device.

The condition of your system's signal connections can significantly affect measurement accuracy and repeatability. Improperly mated connections or dirty, damaged or worn connectors can degrade measurement performance. Clean out any loose, dry debris from connectors with clean, low-pressure air (available in spray cans from office supply stores).

If deeper cleaning is necessary, use lint-free swabs and isopropyl alcohol to gently clean inside the connector barrel and the external threads. Do not mate connectors until the alcohol has completely evaporated. Excess liquid alcohol trapped inside the connector may take several days to fully evaporate and may degrade measurement performance until fully evaporated.

Tighten all SMA connections to 5 in-lb max (56 N-cm max)

Signal Connections

1 **RF IN** This port accepts input signals from 1 MHz to 3.9 GHz to the downconverter. The connector is N-type female. The nominal input impedance is 50 Ω. Maximum input power is +27 dBm.

2 **IF OUT** This port outputs the 70 MHz IF signal from the downconverter. The connector is SMA female. The nominal output impedance is 50 Ω.

3 **REF OUT** This port outputs the internal 10 MHz or 100 MHz reference clock. The connector is BNC female. This port is AC-coupled with a nominal output impedance of 50 Ω.

4 **REF IN** This port accepts an external 10 MHz reference signal, allowing an external source to synchronize the internal reference clock. The connector is BNC female. This port is AC-coupled with a nominal input impedance of 50 Ω. Maximum input power is +13 dBm.

5 Indicator LEDs

The SC5304A provides visual indication of important modes. There are two LED indicators on the unit. Their behavior under different operating conditions is shown in Table 1.

Table 1. LED indicator states.

LED	Color	Definition
STATUS	Green	“Power good” and all oscillators phase-locked
STATUS	Red	One or more oscillators off lock
STATUS	Off	Power fault
ACTIVE	Green/Off	Device is open (green) /closed (off) , this indicator is also user programmable (see register map)
ACTIVE	Orange	User initiated standby mode

6 USB Communication Connection

The SC5304A communicates with a host computer through a standard Type-B USB connector located on the rear panel of the unit. SignalCore provides a 79” (2 meter) Type-A to Type-B USB cable in the shipping kit. Cables as long as 197” (5 meter) may be used in accordance with the USB 2.0 specification, with a maximum round-trip delay of 1.5 μs (including hubs) between the host computer and the device. Maximum per cable delay is set by the specification at 5.2 nanoseconds per meter.

7 Power Switch

This switch turns the unit on and off. The switch is SPDT, breaking both the line and neutral legs of the circuit for added safety. Protective earth grounding is provided through the center pin and is bonded to both the SC5304A enclosure and the internal AC-DC power supply.

8 Fuse

Never access the fuse with the unit powered on. Before attempting to remove the fuse for any reason, switch off power to the unit and disconnect the power cord.

The fuse holder located on the rear panel of the unit contains a user-replaceable fuse for protection against shorts and surges. To access the fuse, place a flat blade screwdriver in the slot on the face of the fuse holder and turn counter-clockwise 90 degrees. The center will pop out slightly, allowing removal of the fuse clip. Remove the fuse from the fuse clip and install a new one in its place. Place the clip back into the holder. Insert a flat blade screwdriver into the slot and push in with a slight pressure until the clip is flush with the fuse holder body. Turn clockwise 90 degrees to lock the fuse clip in place.

Replace fuse with same type and rating. The SC5304A fuse type and rating are:

[5x20 mm glass, 500 mA, 250 VAC, Time Lag (T)]

9 Power Connection

Power is provided to the device through a standard IEC C-14 style power entry connector as shown in Figure 2. An internal switching power supply allows for a range of input voltages, from 88-264 VAC, with line frequencies ranging from 47-63 Hz. A power cable for use in North America is provided in the shipping kit. Individual customers may have to replace this cable with one appropriate for use in their country.

Ensure that a properly grounded power cable is used at all times. Bypassing the ground connection can increase the risk of serious injury or death.

SC5304A-THEORY OF OPERATION

Overview

The SC5304A operates on the principle of heterodyning, a process whereby an incoming RF signal is mixed with specific oscillator frequencies in stages, producing both sum and difference frequency products. At each stage the summed frequency product (or image) is removed through low-pass filtering, allowing the difference frequency product to continue through the signal path. Repeating this process several times using carefully selected local oscillators (LOs) and well-designed band-pass filtering, the original signal is translated, or "down-converted" in frequency low enough for inexpensive digitizers to acquire the signal with reasonable bandwidth. The resultant output signal of a heterodyne downconverter is known as the intermediate frequency (IF). Using a tunable LO as the first mixing oscillator allows the downconverter to translate a broad range of frequencies to a common IF output. When combined, a tunable LO and extraction of the lower mixed frequency product creates an important and useful variant of the heterodyne process known as superheterodyning.

The SC5304A is a three-stage superheterodyne downconverter that delivers superior image rejection over single stage conversion and offers both high signal-to-noise dynamic range and high spurious-free dynamic range. The RF input ranges from 1 MHz to 3.9 GHz, and the IF output is fixed at 70 MHz. When the input frequency is lower than the intermediate frequency, the device technically behaves as an upconverter. The SC5304A up-converts when the input frequency ranges from 1 MHz to 70 MHz. The converted spectrum polarity may be inverted or non-inverted by programming the device accordingly.

Fundamentally, each conversion stage consists of a frequency mixer that mixes two input signals and producing a wanted third. The *wanted* third component is selected, via a frequency filter, among other signals generated in the mixing process. The three primary components of the signals in each conversion mixer are commonly known as the local oscillator (LO), radio frequency (RF), and the intermediate frequency (IF) as shown in Figure 3.

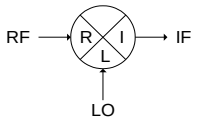


Figure 3. Frequency conversion stage using a mixer.

Where R represents the RF component, L represents the LO component, and I represents the IF component. The LO is resident in the downconverter and is either frequency tunable or fixed in frequency depending on the stage.

The first IF stage is an up-conversion stage - all input signals are converted to an IF higher than the highest input frequency specified. The second and third stages successively convert this high first IF down to the final IF of 70 MHz. Having a high first IF allows the downconverter to achieve very high image rejection ability. This image-free architecture achieves high image rejection without the need for sharp cut-off pre-select band-pass filters. Having high image rejection makes the SC5304A suitable for

applications such as spectral monitoring, broadband spectral analysis, and others where the spectral environment cannot be controlled.

The SC5304A exhibits very low phase noise of -107 dBc/Hz at 10 kHz offset on a 1 GHz RF carrier with a typical noise floor of -150 dBm/Hz. The noise floor can be further reduced below -165 dBm/Hz by enabling the internal preamplifier. With gain control between -60 dB to +50 dB, a measurement signal-to-noise dynamic range greater than 180 dB is achievable. Using high reverse isolation devices and sharp cutoff filters, LO leakages and other spurious contents at the input connectors are well below -120 dBm. Inter-stage LO leakages are also kept very low through sophisticated circuit and shielding design to ensure that spurious in-band signals remain less than -80 dBc. The excellent spurious free dynamic range is achieved using low noise linear amplifiers, low loss mixers, and high performance solid state attenuators. State-of-the-art solid state attenuators have improved linearity over earlier designs. Their attenuation level changes settle under a microsecond, and for applications that involve frequent range changing, they offer a vastly superior lifetime over mechanical attenuators.

The real-time bandwidth is shaped primarily by the final 70 MHz IF surface acoustic wave (SAW) filter. The final IF filter has two programmatically selectable paths, switching either between two filter paths with different bandwidths or between one filter and one bypass (no filter) path. Filters in the first and second IF stages are not as selective as the final IF filter but they ensure good isolation between local oscillators (LO). Keeping each LO isolated helps to suppress unwanted spurious signals.

Frequency accuracy is provided by an onboard 10 MHz temperature compensated crystal oscillator (TCXO) which can be phase-locked to an external reference source if required, and it is recommended to do so in applications that may require a more stable and accurate base reference.

Signal Path Description

Figure 4 depicts an overall block diagram of the SC5304A. Starting from the upper left, the RF input of the SC5304A is AC coupled, followed by an elliptic low-pass filter which has a sharp cut-off frequency slope to ensure the images and unwanted frequencies are well suppressed. Next, a bypass switch enables or disables the internal preamplifier in the path of the RF signal, directly after the input filter. The advantage of placing the amplifier before the attenuators is to increase the downconverter sensitivity when the preamplifier is selected. This switch is programmatically controlled and can be toggled as required, enabling the preamplifier to boost input signals of very small amplitude. Due to losses in the attenuators, the noise figure of the system is proportional to their accumulated losses if the attenuators were placed before the amplifier. The trade-off for better sensitivity is the lack of attenuation adjustment for larger signals when the amplifier is enabled. The user will need to provide good judgment when enabling the preamplifier.

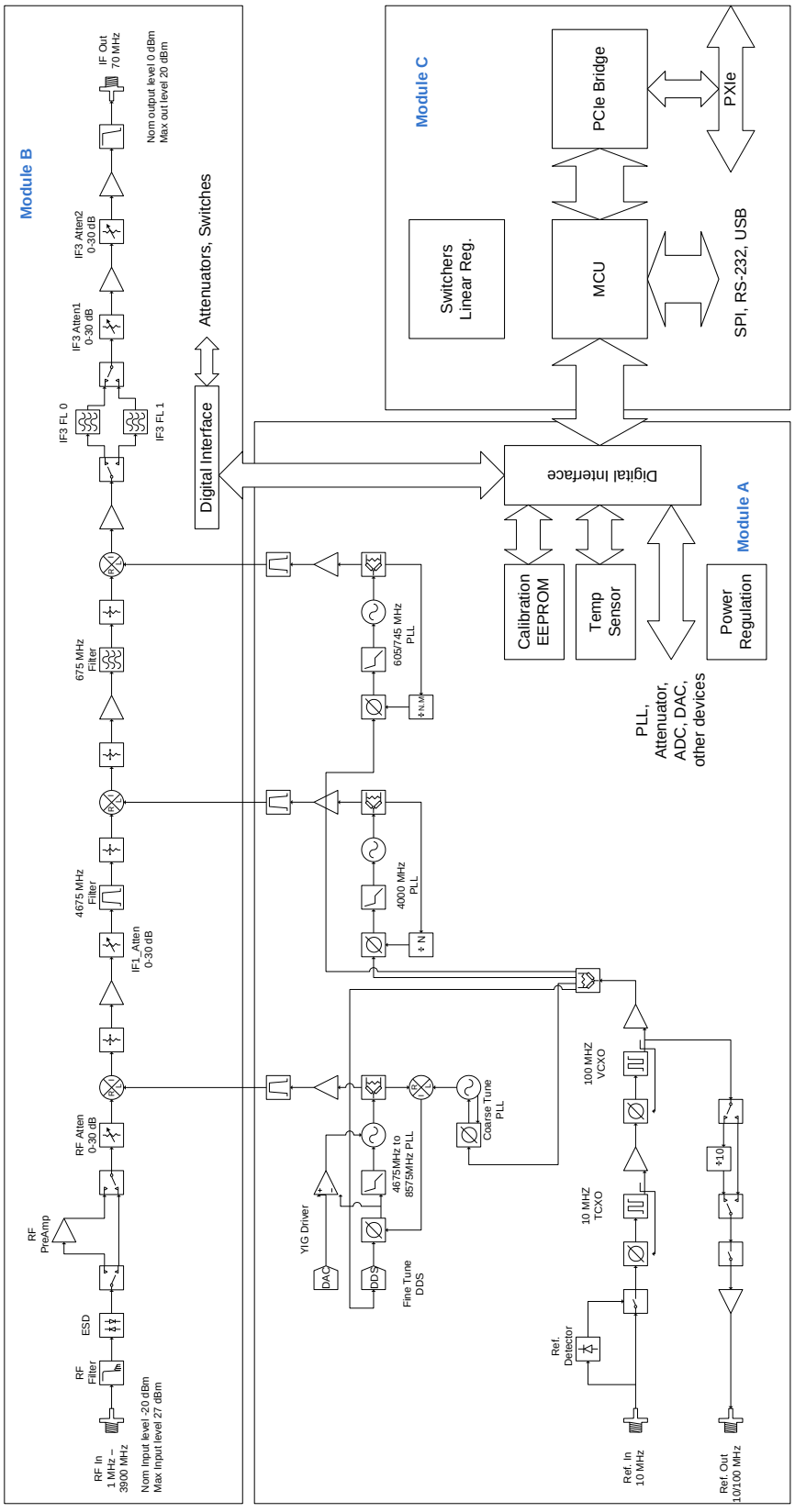


Figure 4. Block diagram of SignalCore 3.9 GHz downconverters.

The RF attenuator, with a 0 to 30 dB attenuation range and attenuation steps of 1 dB, is located between the preamplifier and the first mixer. The RF attenuator is used to set the signal amplitude to a user-defined level at the mixer when the RF input level is higher than that level. This attenuator is adjusted to obtain the required distortion levels. Lowering the RF level with the attenuator at the mixer operates the device in a more linear region. However suppressing the RF level too low before the mixer reduces the signal-to-noise ratio, so the user must set this level to compromise between noise and linearity. The RF signal is mixed with the first local oscillator, LO1, and the difference component is selected as the wanted intermediate frequency (IF).

The first IF stage after the RF mixer, referred to as IF1 in the programming section, is heavily filtered and carefully amplified to maintain the best compromise between signal dynamic range and linearity. The filters provide isolation between the first and second stage mixers to reduce in-band inter-modulation spurious signals from the mixing of high order harmonics of the IF and LO frequencies. The filters in this stage also suppress the mixer LO leakage. If not filtered, LO leakages can potentially cause saturation in the preceding stages of the signal path and degrade the linearity performance of the device. There is an adjustable attenuator following the output of the first mixer, which is used to suppress leakages from LO1 that appear in-band when the downconverter is tuned to frequencies less than the bandwidth of the device. For example, if the bandwidth of the system is 20 MHz, LO1 leakage will appear in-band if the frequency is tuned below 20 MHz. Technically, LO leakages should not appear in-band until the device is tuned below 10 MHz, but the non-ideality of the filter allows sufficient leakage at higher frequencies. Setting this attenuator will attenuate both the IF1 signal and the LO1 leakage, making the device respond more linearly. As always, the compromise is that the SNR will degrade. The LO1 leakage signal will appear as DC when the IF is digitized and converted to baseband. By design, setting the IF frequency at 4675 MHz allows sufficient frequency separation from the highest RF frequency so that the IF1 filter, despite its non ideal roll-off response, can suppress the RF signal by more than -100 dB.

The first IF is then down-converted to the second IF of 675 MHz by mixing with the second LO (LO2). Similarly, as with the first IF section, the second IF (IF2) section is also well filtered and amplified. Keeping isolation between the second and third mixers is important to ensure spurious signals generated within the device are kept significantly low when compared to the primary signal of interest.

Finally, the second IF is converted to the third and final IF by mixing with the third LO (LO3). Located in this stage are the primary band-pass filters that define the bandwidth of the device. The final IF filters are selectable between two filters of different bandwidths centered at 70 MHz. The standard bandwidths for these filters are 5 MHz, 10 MHz, 20 MHz, and 40 MHz. These surface acoustic wave (SAW) filters provide excellent filter response. The user may choose to use one of these two filter paths as a bypass, that is, no band-pass filter in the path. One reason for bypassing the final IF filter is to improve the group delay through the device; with the filters enabled the delay is approximately 1 μs. Bypassing the filter reduces the delay to about 100 ns, which may be preferred in some applications.

Following the band-pass filter are the IF attenuators, IF3_Atten1 and IF3_Atten2. These IF attenuators control the IF gain of the device and set the desired output IF level at the IF output port. The recommended output level is 0 dBm. However, the level may be set to other values that suit the particular application. Finally, a low pass filter suppresses the harmonics of the IF signal. It is important that the IF harmonics are kept as low as possible because they appear in-band as higher order images

when digitized. The harmonics are typically below 90 dBc at the IF. In applications where this may not be acceptable, external analog filtering is recommended.

Local Oscillator Description

The signal path circuit is separate from the local oscillator generation circuits to maximize isolation between the RF/IF signals and the local oscillators, except for the LO injection paths into the mixers. Although the both circuits reside within the same module, well-designed shielding and circuit layouts ensures leakages between them are keep to a minimum.

The first local oscillator, LO1, is an agile, tunable phase-lock synthesizer. The synthesizer tunes from 4675 MHz to 8575 MHz, a tuning range of 3900 MHz. The minimum step size is 1 Hz, and is accomplished through a multiple phase-locked loop and DDS hybrid architecture. The use of a hybrid tuning architecture is important for improved phase noise and improved close-in phase spurious responses. Operating LO1 at such high frequencies internally to obtain a 1 MHz to 3.9 GHz RF range requires that the phase noise at these frequencies is sufficiently low so that the converted RF signal phase noise is not degraded significantly. For example, to downconvert a 100 MHz RF signal, LO1 is tuned to 4775 MHz, which is about 48 times higher in frequency than the input frequency. To further ensure phase noise remains low farther away from the carrier, especially at 100 kHz and 1 MHz offsets, a YIG oscillator is used. It important to realize that having a phase noise “plateau” out to several tens of MHz, which is a very common phenomenon with VCO-based synthesizers, is not acceptable for many applications.

Another reason for a hybrid tuning architecture is to reduce the phase spurs associated with phase-locked loops. A simple fractional PLL may provide resolution to 1 Hz, but it cannot provide 1 Hz frequency tuning steps with low fractional phase spurs. By using two DDS circuits to provide the 1 Hz tuning steps and mathematically ensuring that DDS-generated spurs are suppressed within the architecture, LO1 is made to fine tune to exact frequencies, that is, the frequency synthesized is an exact integer multiple or division of the reference signal.

The second local oscillator, LO2, is fixed at 4.0 GHz, synthesized using an integer PLL and a fixed narrow tune VCO with very low phase noise. The typical raw phase noise of the second stage oscillator is less than -150 dBc/Hz @ 1 MHz offset. LO2 phase noise contribution to the overall phase noise of the device is less than 1 dB. LO1 dominates the phase noise of the device.

The third local oscillator, LO3, is synthesized using a fractional PLL and has phase noise lower than both LO1 and LO2, and is switchable between two frequencies: 605 MHz and 745 MHz, the later frequency being the default. Both of these frequencies will set the output IF center frequency at 70 MHz. However, at the default frequency the final 70 MHz IF output spectral polarity is the same as that of the input RF, whereas the 605 MHz frequency will create an inverted IF spectrum. If LO3 is set to 605 MHz by calling the **sc5304a_SetIfInversion** function or register, the IF output spectral content will be inverted with respect to the input RF spectrum. See Figure 5 for a graphical representation of this process.

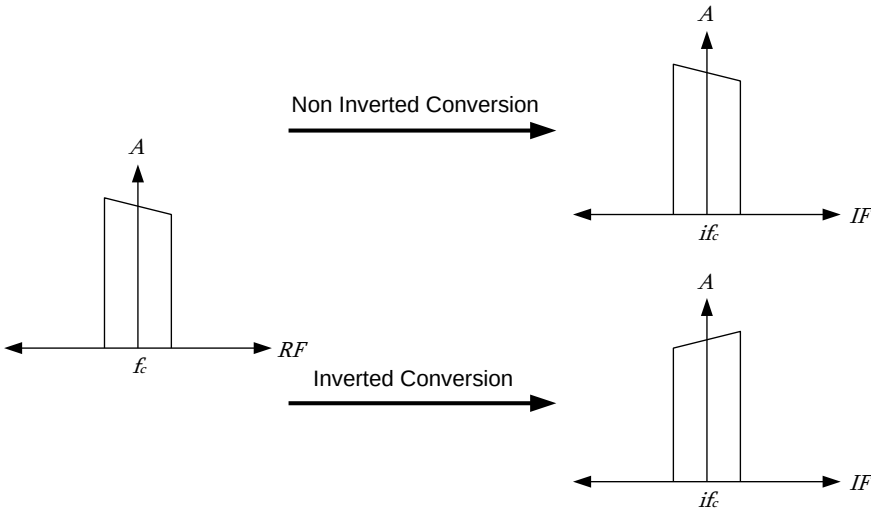


Figure 5. Graphical representation of IF inversion.

Inverted spectral conversion is convenient for digitizers that sample the IF in the even Nyquist zones because it eliminates the need to perform digital inversion of the acquired spectrum.

All local oscillators are phase-locked to an internal 100 MHz voltage controlled crystal oscillator (VCXO), which sets their close-in phase noise performance. The 100 MHz VCXO is in turn phase-locked to the internal 10 MHz TCXO for frequency accuracy and stability. For better frequency accuracy and stability than the TCXO onboard the SC5304A, or for frequency synchronization, the user can programmatically set the device to phase lock the TCXO to an external 10 MHz reference source by programming the REFERENCE_SETTING register. It is important to note that the TCXO will only attempt to lock to an external source if one is detected. A typical external reference source minimum level of -10 dBm is required for detection to be successful. A reference source level of 0 dBm to +3 dBm is recommended for normal operation. The reference source is fed into the device through the **"ref in"** port. The device can also export a copy of its internal reference through the **"ref out"** port. The output reference frequency is selectable for either 10 MHz or 100 MHz output. By default, routing of the reference signal to the "**ref out**" port is disabled. It can be enabled by programming the REFERENCE_SETTING register. This reference frequency is sourced from the internal 100 MHz OCXO, and the default output selection is 10 MHz, which is divided down from the 100 MHz VCXO. The output reference level is typically +3 dBm.

Frequency Tuning Modes

Tuning of SC5304A superheterodyne downconverter is accomplished through the tuning of LO1. LO1 has two sets of control parameters that can be explored to optimize the device for any particular application. The first set of parameters, TUNE SPEED, set the tuning and phase lock time as the frequency is changed. TUNE SPEED consists of two modes - Fast Tune mode and the Normal mode; both of these modes directly affect the way the YIG oscillator is configured. The Fast Tune mode deactivates a noise suppression capacitor across the tuning coil of the YIG oscillator, and doing so increases the rate of current flow through the coil, correspondingly increasing the rate of frequency change. In Normal mode the capacitor is activated, slowing down the rate of frequency change. The advantage of activating the capacitor is that it shunts the noise developed across the coil, decreasing close-in phase noise. Refer to Appendix A for specifications regarding tuning speed.

The other set of control parameters, FINE TUNE, sets the tuning resolution of the device. There are three modes: 1 MHz, 25 kHz, and 1 Hz tuning step sizes. The first two modes use only fractional phase detectors to tune the frequency of the LO1 synthesizer, while the third mode enables the DDS to provide 1 Hz resolution. The PLL-only modes (1 MHz and 25 kHz) provide the ability to realize exact frequencies with tuning as fine as 25 kHz. Use of these modes offers several advantages - lower phase spurs and less computational burden to set a new frequency. These modes have the lowest phase spurious signals, below the levels published in the product specification. The DDS mode also tunes to exact frequencies, however it requires many more computing cycles and additional register-level writes in order to set a new frequency. Comparing times, the device requires up to 175 microseconds to compute and change to a new frequency in PLL only modes, but requires up to 350 microseconds in the DDS tuning mode. At first glance it may seem that these differences would directly impact frequency tuning times. However, tuning times are predominantly set by the physical parameters of the YIG oscillator. Computation and register writes typically account for less than 25% of the total tune time of a 10 MHz step change in frequency.

It is important to note that although the synthesized frequencies are exact frequencies, there are observable random phase drifts in the downconverted signals. These drifts are due to PLL non-idealities rather than a frequency error in the DDS tuning circuit. Having exact frequency synthesis is important for many applications. Published phase noise and spurs specifications are based on the 1 Hz (DDS) mode.

Setting the SC5304A to Achieve Best Dynamic Range

When discussing dynamic range, there are two distinct quantities which are specified. First is the compression-to-noise density (per Hz) dynamic range, commonly referred to in SignalCore literature as the signal-to-noise ratio dynamic range (SNRDR). Second is the third order spurious free dynamic range, commonly known as the SFDR. In traditional radio terminology, the SFDR strictly refers to the third order effects of nonlinearity whose products are generally close to the carrier signal and are very difficult to filter out. In analog to digital conversion, the SFDR term takes on a different definition, referring to the ratio of the input signal strength to all the spurious products appearing within the Nyquist band. These later spurious signals may be caused by harmonics, inter-modulation, and digital quantization.

These two dynamic ranges are instantaneous, in that the signal and the noise or spurs are observed at the same time. On the other hand, the measurement dynamic range, specifically referring to SNRDR, is not instantaneous. The user may enable RF attenuation to receive signals levels much greater than the instantaneous compression point, or enable the preamplifier to detect signals below the instantaneous noise density level. The measurement dynamic range is thus much greater than the instantaneous equivalent.

The SC5304A was designed with a focus on a high dynamic range, not just low in noise or just having high compression points. It was designed as a receiver for signal analyzers, which require that it handle larger signals well. For weak signals, the RF preamplifier should be enabled. The design ensures the SFDR dynamic range specification is met when the RF signal level at the input mixer is -20 dBm and the IF level is at 0 dBm. This requires a total IF attenuation of 10 dB for a typical device gain of 30 dB (preamplifier disabled). This setting is typical for broadband signals with more than a few MHz of real-time bandwidth.

For applications where the SNR must be maximized, such as examining the close-in characteristics of a sine tone, the input mixer should be set to accept 0 dBm power and the IF set at 0 dBm or higher. This is a likely setting for making phase noise measurements of an RF signal (assuming the specified phase noise of the SC5304A is low enough for measuring that particular signal). It is important to first set the necessary attenuators before injecting a 0 dBm level signal to the mixer, otherwise heavy saturation of the mixer or the output amplifiers may cause degradation or even possible failure of the receiver over time.

The SC5304A is designed for a nominal output IF level of 0 dBm, ensuring the IF signal is about 3-4 dB below the full-scale value of many 50 Ω analog-to-digital data converters (ADCs). Depending on the application, the user will need to set the appropriate gain of the device (via attenuation), and hence the output level, to suit the particular application. For broadband signals, it is recommended that the IF output level be about 7 dB below the full-scale value of a digitizer because of possible high crest factors that may saturate the digitizer. For sine-tone or narrowband application, the output IF level should be about 3 dB below full-scale of the digitizer to maximize its signal-to-noise dynamic range.

SignalCore provides a simulation tool that mimics the behavior of the SC5304A. The user may run the simulator to get an understanding of what the parameters need to be set on the downconverter to achieve certain performance. Additionally, the function **sc5304a_CalcAutoAttenuation** helps the user obtain the necessary attenuator parameters to setup the device up for best compromise of linearity and noise performance for a given set of input and output parameters.

There is a programmable attenuator in the first IF section, IF1_Atten, that can be used to improve linearity in general. The primary use of this attenuator is to suppress the LO1 leakage in the IF band when the downconverter is tuned below the bandwidth frequency. This in-band leakage affects the linearity of the device as it may inter-modulate with the IF signal to produce third order spurious products. The level of the leakage is equivalent to a typical -25 dBm RF signal at the mixer. The user should set 5 dB to 10 dB OF attenuation when operating at these low frequencies.

Operating the SC5304A Outside Normal Range

The SC5304A is capable of tuning below 1 MHz and above 3.9 GHz. These frequencies lie outside of the specification range and performance will be degraded if operated in these outer margins. However, for some applications, the reduced dynamic range or elevated spurious levels in these ranges may not pose an application concern. The lowest tunable frequency is 0 MHz (DC). However, for input frequencies below 1 MHz, the AC coupling capacitors in the circuit limit/attenuate the signal significantly. On the upper end of the spectrum, the input low-pass filter will attenuate the signal rapidly as the frequency increases above 3.9 GHz. Calibration stored on the device EEPROM does not account for these out of range frequencies, so applying any correction using the stored calibration is not valid.

SC5304A Programming Interface

Device Drivers

The SC5304A is programmatically controlled by writing to its set of configuration registers, and its status read back through its set of query registers. The user may choose to program directly at the register level or through the API library functions provided. These API library functions are wrapper functions of the registers that simplify the task of configuring the register bytes. The register specifics are covered in the next section. Writing to and reading from the device at the register level through the API involves calls to the **sc5304a_RegWrite** and **sc5304a_RegRead** functions respectively. Writing to the device at the raw register level is independent of the host operating system and is suitable for embedded applications.

There are many USB drivers available that can be used to directly access the registers of the device - WinUSB, libusb-1.0, libusb-win32, libusbx, and NI-VISA are a few examples. The SC5304A API is provided as a dynamic linked library (*sc5304a.dll*) or shared library (*libsc5304a.so*). This API is based on the libusb-1.0 driver and is available for both Windows and Linux operating systems. For more information on the libusb driver, visit www.libusb.org. For the Windows platform, libusb-1.0 relies on WinUSB at the kernel level. For libusb-1.0 to function properly on Windows platforms, WinUSB must be installed on the host.

For LabVIEW support on Windows operating systems, the LabVIEW API *sc5304a.llb* is provided on the software installation CD and is based on the NI-VISA driver platform. For Windows Vista or later operating systems, the NI-VISA driver uses WinUSB at the kernel level. For these systems, installing just the NI-VISA driver will provide support for both NI-VISA and libusb-1.0 drivers. However on Windows XP systems, NI-VISA does not rely on WinUSB, so the user must choose to install the NI-VISA driver for LabVIEW programming, or the libusb-1.0 driver for C/CC++, VBASIC, and other languages that can call C/C++ style DLLs. The user should only install one driver on the host to reduce driver load confusion by the operating system.

For other operating systems or embedded systems, users will need to access the registers through their own proprietary USB driver or through one of the drivers mentioned above. Should the user require additional assistance in writing an appropriate API other than that provided on the software installation CD, please contact SignalCore.

The SC5304A USB Configuration

The SC5304A USB interface is USB 2.0 compliant running at Full Speed, capable of 12 Mbits per second. It supports 3 transfer or endpoint types:

- Control Transfer
- Interrupt Transfer
- Bulk Transfer

The endpoint addresses are provided in the C-language header file and are listed below.

```
// Define SignalCore USB Endpoints
#       define  SCI_ENDPOINT_IN_INT         0x81
#       define  SCI_ENDPOINT_OUT_INT        0x02
#       define  SCI_ENDPOINT_IN_BULK        0x83
#       define  SCI_ENDPOINT_OUT_BULK       0x04

// Define for Control Endpoints
#       define  USB_ENDPOINT_IN             0x80
#       define  USB_ENDPOINT_OUT            0x00
#       define  USB_TYPE_VENDOR             (0x02 << 5)
#       define  USB_RECIP_INTERFACE         0x01
```

The buffer lengths are sixty-four bytes for all endpoint types. The user should not exceed this length or the device may not respond correctly. Furthermore, all registers require five bytes or less with the exception of registers 0x24 (FETCH_CAL_EEPROM_BULK) and 0x25 (FETCH_USER_EEPROM_BULK). These two registers return sixty-four bytes of valid data upon executing a device read.

Writing the Device Registers Directly

Device commands for the SC5304A vary between two bytes and five bytes in length. The most significant byte (MSB) is the command register address that specifies how the device should handle the subsequent configuration data. The configuration data likewise needs to be ordered MSB first, that is, transmitted first. For configuration commands, an output buffer of five bytes long is sufficient. To ensure that a configuration command has been fully executed by the device, reading a few bytes back from the device will confirm a proper command write because the device will only return data upon full execution of a command. A map of the configuration registers for the SC5304A is provided in Table 2. Details for writing the registers are provided in the next section.

Reading the Device Registers Directly

Valid data is only available to be read back after writing one of the query registers. With the exception of registers 0x24 (FETCH_CAL_EEPROM_BULK) and 0x25 (FETCH_USER_EEPROM_BULK, where sixty-four bytes of valid data are returned, only the first two bytes of the device IN endpoints contain valid data. When querying the device, the MSB is returned as the first byte. A map of the query registers for the SC5304A is provided in Table 3. Details for reading the registers are provided in the next section.

Using the Application Programming Interface (API)

The SC5304A API library functions make it easy for the user to communicate with the device. Using the API removes the need to understand register-level details - their configuration, address, data format, etc. Furthermore, the user does not need to understand the different transfer types of the USB interface. For example, to obtain the device temperature the user simply calls the function **sc5304a_GetDeviceTemperature**, or calls **sc5304a_SetFrequency** to set the device frequency. The software API is covered in detail in the "Software API Library Functions" section.

SETTING THE SC5304A - WRITING CONFIGURATION REGISTERS DIRECTLY

Configuration Registers

The configuration registers to control the SC5304A are summarized in Table 2. The size of a command is based on the sum of the register address byte and the number of data bytes. For example, setting the frequency would require five bytes with the first byte being the register address 0x10. The user may send more data bytes than required for a command without incurring a device error. “Extra” bytes will be ignored.

Table 2: Configuration registers.

Register (Address)	Data Bytes	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default
INITIALIZE (0x01)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Mode	0x00
SET_SYSTEM_ACTIVE (0x02)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable SYS LED	0x00
POWER_SHUT_DOWN (0x05)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	0x00
RF_FREQUENCY (0x10)	[7:0]	Frequency Word [7:0]								0x00
	[15:8]	Frequency Word [15:8]								0x00
	[23:16]	Frequency Word [23:16]								0x00
	[31:24]	Frequency Word [31:24]								0x00
ATTENUATOR_SETTING (0x11)	[7:0]	Attenuator Value								0x00
	[15:8]	Attenuator								0x00
RF_PREAMPLIFIER_SETTING (0x12)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	0x00
RF_MODE_SETTING (0x13)	[7:0]	Open	Open	Open	Open	Open	Fast Tune Enable	Fine Tune		0x00
IF_BAND_SELECT (0x15)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Band	0x00
REFERENCE_SETTING (0x16)	[7:0]	Open	Open	Open	Open	Reserv	100 MHz Out Sel	Ref Out Enable	Lock Enable	0x00
REFERENCE_DAC (0x17)	[7:0]	DAC word [7:0]								0x00
	[15:8]	DAC word [15:8]								0x00
IF_INVERT_SETTING (0x1D)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Enable	
WRITE_USER_EEPROM (0x23)	[7:0]	EEPROM DATA [7:0]								
	[15:8]	EEPROM Address [7:0]								
	[23:16]	EEPROM Address [15:8]								
PHASE_SETTING (0x32)	[7:0]	Units Value[7:4]				Tenths Value				0x00
	[15:8]	Open	Open	Units Value [13:8]						0x00

Tuning the RF Frequency

The frequency of the first local oscillator (LO1) is set by writing the RF_FREQUENCY register (0x10). This register requires the address byte plus four data bytes, these data bytes being the bytes comprising an unsigned 32-bit integer. The data bytes contain the frequency tuning word in Hertz. For example, to tune to a frequency of 2.4 GHz, the data word would be d2400000000 in decimal or 0x8F0D1800 in hexadecimal. Breaking this command down to the five bytes sent via the USB interface, the command bytes would be [0x10] [0x8F] [0x0D] [0x18] and [0x00], with [0x10] being the first byte sent.

Changing the Attenuator Settings

The ATTENUATOR_SETTING (0x11) register has two data bytes needed to set the value of a specific attenuator. The MSB sets the target attenuator, and the least significant byte (LSB) contains the attenuation value. The MSB values and corresponding attenuator locations are as follows:

MSB value	Attenuator
0x00	IF3_ATTEN2
0x01	IF3_ATTEN1
0x02	RF_ATTEN
0x03	IF1_ATTEN

The LSB contains the attenuation value in 1 dB steps for the attenuator specified in the MSB. For example, to set the RF attenuator to 15 dB, the command bytes would be [0x11] [0x02] [0x0F].

Enabling and Disabling the RF Preamplifier

The RF_PREAMPLIFIER_SETTING (0x12) register has one data byte that enables or disables the RF preamplifier. It is recommended that the preamplifier only be enabled when the RF input signal is less than or equal to -30 dBm. Enabling the preamplifier increases the receiver sensitivity for low-level signals. Setting the LSB of the data byte high or low will enable or disable the RF preamplifier, respectively. For example, to turn on the preamplifier the command bytes would be [0x12] [0x01].

Changing the RF Synthesizer Mode

The RF_MODE_SETTING (0x13) register has one data byte that provides two tuning modes for the device: Fast Tune and Fine Tune. By default the Fast Tune mode is disabled (Normal mode). Asserting high bit 3 of the data byte will enable Fast Tune mode. Fast Tune enables the device to achieve faster lock and settling times between frequency changes. Please refer to Appendix A for more information regarding Fast Tune mode. The second mode, Fine Tune mode, has three options: 1 MHz (PLL), 25 kHz (PLL), and 1 Hz (DDS). Selection of these options requires setting the first two bits of the data byte to 0, 1, and 2 respectively. See the **Frequency Tuning Modes** section for more information. For example, to set the device for Fast Tune and a 1 Hz tuning step resolution, the command bytes would be [0x13] [0x06].

Selecting the IF Filter Path

The IF_FILTER_SELECT (0x15) register has one data byte that selects between two installed IF filters; IF3_FILTER0 and IF3_FILTER1. Setting bit 0 high will select IF_FILTER1. The exact bandwidths of the filters depend on the available installed options and are stored in the device calibration EEPROM.

Setting the Reference Clock Behavior

The REFERENCE_SETTING (0x16) register has one data byte which sets the reference clock behavior of the device. The default state of this register is 0x00, which disables the export of the internal reference clock, and disables phase locking to an external source. Asserting bit 0 high enables the device to lock to an external clock source. However, the device will not attempt to phase lock until it successfully detects the presence of a clock source at the "**ref in"** port. Asserting bit 1 low disables export of the internal clock. Asserting bit 1 high enables the device to export a 10 MHz signal through the "**ref out**" port. Asserting high bit 1 and bit 2 exports a 100 MHz signal.

Adjusting the Reference Clock Accuracy

The frequency precision of the SC5304A's 10 MHz TCXO is set by the device internally. The device writes the factory calibrated value to the reference DAC on power-up. This value is an unsigned 16-bit number stored in the EEPROM (see the calibration EEPROM map). The user may choose to write a different value to the reference DAC by accessing the REFERENCE_DAC (0x17) register. This register has two data bytes to hold the 16-bit word.

Setting Spectral Inversion in the IF

The default IF spectral polarity is the same as that of the RF input. However, should there be a need to invert the IF spectrum with respect to the RF spectrum, the register IF_INVERT_SETTING (0x1D) is used for that purpose. This register contains one data byte. Setting bit 0 high will enable inversion.

Storing Data into the User EEPROM Space

There is an on-board 16k byte EEPROM available to the user to store user data information such as user calibration, settings, etc. Writing to the user accessible EEPROM space is accomplished through the register WRITE_USER_EEPROM (0x23). This register has three data bytes: bytes 2 and 1 contain the address of the EEPROM; byte 0 is the byte value to be written. **NOTE:** The user must add a 5 millisecond delay between consecutive writes. There is no delay required in read mode. For example, to write 123 to address 1234 of the user EEPROM the command bytes would be [0x23] [0x04] [0xD2] [0x7B].

Setting the Phase of the IF Signal

When the device is tuned to a fixed RF frequency, some applications may need to change the phase of the down-converted signal for various reasons. The IF output phase can be programmatically adjusted in 0.1 degree increments from 0 to 360 degrees. Changing the phase is accessed through the register PHASE_SETTING (0x32), which has two data bytes. The first 4 bits contain the tenths value, while bits [13:4] hold the units value.

QUERYING THE SC5304A: WRITING REQUEST REGISTERS DIRECTLY

The request set of registers are shown below in Table 3. To obtain requested data from the device requires two steps:

1) Write the request register via the OUT endpoint with the appropriate data if any required.
2) Read the USB IN endpoint to obtain the data requested.

Table 3: Query registers.

Register (Address)	Data Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default
FETCH_DEVICE_STATUS (0x18)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open	0x00
Read Byte 1	[7:0]	Device status [7:0]								
Read Byte 0	[15:8]	Device status [15:8]								
FETCH_TEMPERATURE (0x19)	[7:0]	Open	Open	Open	Open	Open	Open	Open	Open	0x00
Read Byte 1		Temp data [7:0]								
Read Byte 0		Open	Open	Sign	Temp data [12:8]					
FETCH_CAL_EEPROM (0x20)	[7:0]	EEPROM Address [7:0]								0x00
	[15:8]	EEPROM Address [15:8]								0x00
Read Byte 1		Cal EEPROM byte data								
Read Byte 0		Open	Open	Open	Open	Open	Open	Open	Open	
FETCH_USER_EEPROM (0x22)	[7:0]	EEPROM Address [7:0]								0x00
	[15:8]	EEPROM Address [15:8]								0x00
Read Byte 1		User EEPROM byte data								
Read Byte 0		Open	Open	Open	Open	Open	Open	Open	Open	
FETCH_CAL_EEPROM_BULK (0x24)	[7:0]	EEPROM Start Address [7:0]								
	[15:8]	EEPROM Start Address [15:8]								
Read Byte 63		Cal EEPROM data @ Start Address +63								
:					:					
Read Byte 0		Cal EEPROM data @ Start Address								
FETCH_USER_EEPROM_BULK (0x25)	[7:0]	EEPROM Start Address [7:0]								
	[15:8]	EEPROM Start Address [15:8]								
Read Byte 63		User EEPROM data @ Start Address +63								
:					:					
Read Byte 0		User EEPROM data @ Start Address								

Reading Device Status Data

To obtain the device status, write request register FETCH_DEVICE_STATUS (0x18) with 0x00 in the data byte, followed by querying the two bytes of data at the corresponding USB endpoint. The returned data are summarized in Table 4. It is important to note that the first local oscillator has three phase detectors in the synthesizer, so all three phase detectors must be ANDed to indicate the proper phase-locked status. The three bits that indicate the status of the three phase detectors are [13], [10] and [9].

Table 4. Description of the status data bits.

Bit	Description
[15]	10 MHz TCXO PLL lock status
[14]	100 MHz VCXO PLL lock status
[13]	LO1 PLL Main lock status
[12]	LO2 PLL lock status
[11]	LO3 PLL lock status
[10]	LO1 PLL1 lock status
[9]	LO1 PLL2 lock status
[8]	Reserved
[7]	External reference detected
[6]	Reference output enabled
[5]	Reference lock enabled
[4]	IF3_FILTER1 selected
[3]	RF preamplifier enabled
[2]	Device standby enabled
[1]	Reserved
[0]	Reserved

Reading Temperature Data

To obtain temperature data, write request register FETCH_TEMPERATURE (0x19) with 0x00 in the data bytes, followed by reading two bytes. Once data is received, the two bytes of data need to be processed to correctly represent the data in temperature units of degrees Celsius. Data is returned in the first 14 bits [13:0]. Bit [13] is the polarity bit indicating whether it is a positive (0x0) or negative (0x1) value. The temperature value represented in the raw data is contained in the next 13 bits [12:0]. To obtain the temperature ADC code, the raw data should be masked (logically AND'ed) with 0x1FFF, and the polarity should be masked with 0x2000. The conversion from 12 bit ADC code to an actual temperature reading in degrees Celsius is shown below:

Positive Temperature = ADC code / 32

Negative Temperature = (ADC code – 8192) / 32

It is not recommended to read the temperature too frequently, especially once the SC5304A has stabilized in temperature. The temperature sensor is a serial device located inside the RF module.

Therefore, like any other serial device, reading the temperature sensor requires sending serial clock and data commands from the processor. The process of sending clock pulses on the serial transfer line may cause unwanted spurs on the RF signal as the serial clock potentially modulates the local oscillators. Furthermore, once the SC5304A stabilizes in temperature, repeated readings will likely differ by as little as 0.25 ^{O}C over extended periods of time. Given that the gain-to-temperature coefficient is on the order of 0.1 dB/^{O}C, gain changes between readings will be negligible.

Reading Calibration EEPROM Data

To read a single byte from an address in the device EEPROM write the FETCH_CAL_EEPROM register with the address in the two data bytes, followed by reading back 2 bytes of data. The data is returned on the last byte, byte 1; byte 0 contains invalid data. The EEPROM maximum address is 0x3FFF. Reading above this address will cause the device to retrieve data starting from the lowest addresses. For example, addressing 0x4000 will return data stored in address location 0x0000. The calibration EEPROM map is discussed in detail in the Working With Calibration Data section.

For faster reading of the entire calibration EEPROM, the FETCH_CAL_EEPROM_BULK (0x24) register should be written with the starting address of the 64 byte memory read. Data is return in 64 byte chunks where the first byte corresponds to the starting memory address. There are 237 chunks of 64 byte data.

All calibration data, whether floats or unsigned 32-bit integers, are stored as flattened unsigned 32-bit words. Each data point is comprised of four unsigned bytes, so data must be read back in multiples of four bytes, with the *least significant byte stored in the lower address*. After the data are read back, they need to be un-flattened back to their original type. Since the four bytes constitutes the four bytes of an unsigned 32-bit integer, converting (un-flattening) to an unsigned value simply involves concatenation of the bytes through bit shifting. To convert to floating point representation is a little more involved. First, convert the four bytes into an unsigned 32-bit integer value, and then (in C/C++, etc.) type-cast a float pointer to the address of the value. In C/C++, the code would be float Y = *(float *)&X, where X has been converted earlier to an unsigned integer.

An example written in C code would look something like:

```
byte_value[4]; // read in earlier
unsigned int uint32_value;
float float32_value;

int count = 0;
while (count < 4) {
        uint32_value = unit32_value | (byte_value[count] << (count*8));
        count++;
}

float32_value = *(float *)&uint32_value;
```

Reading User EEPROM Data

Once data has been written to the User EEPROM, it can be retrieved using the process outlined above for reading calibration data, but calling the FETCH_USER_EEPROM or FETCH_USER_EEPROM_BULK registers instead.

WORKING WITH CALIBRATION DATA

The device EEPROM on board has capacity for 16k bytes data. The EEPROM stores both device information and calibration data, which the user may choose to use to correct for conversion gain. Users are not required to use the onboard calibration to compensate for the gain errors associated with temperature, attenuator settings, frequency, pass-band ripple, and filter path selection. Alternatively, users can perform their own system calibration to remove these errors if the unit is integrated into a larger system whose external factors affect the gain significantly. Furthermore, the calibration data provided are raw measured data, and it is in the discretion of the user to decide on the appropriate methods of applying the calibration. For example, the user may choose to fit the measured data to a polynomial and use the polynomial coefficients to compute the necessary correction. Alternatively, the user may choose to perform correction through the use of interpolation. The methods outlined in this section only serve as guides on how to use the calibration data for correction, and these are the methods used by SignalCore in deriving published specifications that indicate the use of calibration. The function **sc5304a_CalcGain** utilizes the methods outlined here and may be used to compute the device gain for any particular setting of the device.

Table 5. EEPROM memory map of device attributes and raw calibration data.

EEPROM ADD (HEX)	# DATA POINTS	# BYTES	TYPE	Array matrix	DESCRIPTION
0X00	1	4	U32	[1x1]	Manufacturing Information
0X04	1	4	U32	[1x1]	Product Serial Number
0X08	1	4	U32	[1x1]	RF Module Number
0X0C	1	4	U32	[1x1]	Product Manufacture Date
0x10	1	4	U32	[1x1]	Last Calibration Date
0X2C	1	4	F32	[1x1]	Firmware Revision
0x30	1	4	F32	[1x1]	LO Hardware Revision
0x34	1	4	F32	[1x1]	SC Hardware Revision
0x50	1	4	F32	[1x1]	Calibration Temperature
0x54	1	4	U32	[1x1]	TCXO DAC Value
0x58	2	8	F32	[1x2]	Reserved
0x64	72	288	F32	[3x24]	YIG Calibration (Reserved)
0x184	1	4	F32	[1x1]	IF Filter 0 Bandwidth
0x188	1	4	F32	[1x1]	IF Filter 1 Bandwidth
0x1A0	24	96	F32	[3x8]	Temperature Coefficients
0x204	153	612	F32	[3x51]	IF3_FILTER0 Response Calibration
0x46C	153	612	F32	[3x51]	IF3_FILTER1 Response Calibration
0x788	1	4	F32	[1x1]	Reserved
0x78C	1	4	F32	[1x1]	IF Invert Gain
0x790	1	4	F32	[1x1]	IF3_FILTER1 Gain
0x798	90	360	F32	[3x30]	IF Attenuator Calibration
0x9F8	1650	6600	R32	[33x50]	RF calibration

Table 5 lists the calibration EEPROM map of the SC5304A, indicating how and where board information and calibration data are stored. Since there are only 16k bytes on the EEPROM, SignalCore recommends that all data be read into host memory on initialization of the device and parsed for further mathematical manipulation. Having it available on host memory at all times during an application will greatly increase the speed of data manipulation. Another recommendation is to store the data to a file and have the application read the file rather than the EEPROM to retrieve data on each execution

because of the relative slow EEPROM read rate. The **sc5304a_ConvertRawCalData** function is helpful to convert EEPROM data to their original format and types.

EEPROM Data Content

The following list describes the data contents of the EEPROM in detail. All addresses shown are the starting offset positions in the EEPROM, and are the starting address for block of data.

Manufacturing Information *(0x00).* This is an unsigned integer value that contains information used by the factory for production purposes.

Product Serial Number *(0x04).* This is an unsigned integer value that contains the SC5304A serial number. It is unique for every product produced. It is used for the purpose of tracking the history of the product.

RF Module Serial Number *(0x08).* This is the serial number of the shielded RF metal enclosure containing the analog and RF circuitry. All calibration data are stored on the EEPROM within the enclosure. Calibration data are written to this EEPROM at the factory are tracked using the RF module serial number for the SC5304A.

Product Manufacture Date *(0x0C).* This is an unsigned integer: byte 3 is the Year, byte 2 is the Month, byte 1 is the day of the month, and byte 0 is the hour of the day.

Last Calibration Date *(0x10).* This is an unsigned integer: byte 3 is the Year, byte 2 is the Month, byte 1 is the day of the month, and byte 0 is the hour of the day.

Firmware Revision *(0x2C).* This is a float 32 value containing the firmware revision.

LO Hardware Revision *(0x30).* This is a float 32 value containing the local oscillator hardware revision.

SC Hardware Revision *(0x34).* This is a float 32 value containing the signal chain hardware revision.

Calibration Temperature *(0x50).* This is a float 32 value containing the temperature at which the device was calibrated.

TCXO DAC Value *(0x54).* This is an unsigned integer containing the value for the reference DAC to adjust the precision of the temperature-compensated crystal oscillator (TCXO).

YIG Calibration (0x64). Data is reserved for device use.

IF Filter Bandwidths (0x184, 0x188). These two float 32 data points contain the filter bandwidths of IF3_FILTER0 and IF3_FILTER1, respectively. These are only available if the product contains non-standard filters, different from those provided with the base product.

Gain/temperature coefficients (0x1A0). This is a 3x8 float matrix, where data is concatenated by rows, that is, data is read back row by row. These coefficients, derived during calibration, are needed to compute for gain as a function of temperature. They are 2nd order polynomial coefficients and are measured over eight different frequencies. See "**Gain Correction**" section for more information on gain

correction factors. **Table 6** is an example of the coefficient data and its format. Variables a_1 and a_2 are the first and second order coefficients.

Table 6. An example of gain-temperature coefficients data and format.

Frequency (MHz)	50.0	250	500	1000	1500	2500	2800	3800
a_1	-0.04500	-0.04800	-0.05600	-0.05000	-0.04500	-0.04800	-0.05600	-0.05000
a_2	-0.00038	-0.00035	-0.00029	-0.00038	-0.00038	-0.00035	-0.00029	-0.00038

IF3_FILTER0 Response Calibration (0x204). This is a 3x51 float matrix, and data is read back row by row. This set of data measures pass-band amplitude variation with respect to the center IF, and phase deviation from linear phase of filter IF3_FILTER0. There are a total of 51 offset frequency points from the center IF frequency measured inside the bandwidth of the filter. Table 7 is an example of the data and format.

IF3_FILTER1 Response Calibration (0x46C). This is a 3x51 float matrix, and data is read back row by row. This set of data measures pass-band amplitude variation with respect to the center IF, and phase deviation from linear phase of filter IF3_FILTER1. There are a total of 51 offset frequency points from the center IF frequency measured inside the bandwidth of the filter. Table 7 is an example of the data and format.

Table 7. Relative IF gain and phase response calibration and format.

Frequency Offset (MHz)	-12	-11.5	...	-5	...	-.5	0	.5	...	5	...	11.5	12
Gain Error	-58.2	-20.6	...	0.9	...	0.2	0.0	0.3	...	-0.2	...	-12.8	-30.4
Phase Error(radians)	.286	.138	...	.053	...	.012	0	.011	...	...	...	-.129	-.134

IF Invert Gain Correction (0x788). This is a float that contains the change in IF gain when the device is switched to invert the IF spectrum. The default gain in the IF is the non-inverted mode gain.

IF3_Filter1 Gain Correction (0x790). This is a float that contains the change in IF gain when the device is switched to the IF3_FILTER1 path. The default gain in the IF is the IF3_FILTER0 path gain.

IF Attenuator Calibration (0x798). This is a 3x30 float matrix, containing the calibrated attenuation values of the three IF attenuators. Data is read in row by row. Each attenuator has 30 attenuation steps, and each row correspond to one attenuator. The first row is the attenuation values of IF3_ATTEN2, the second row contains the values or IF3_ATTEN1, and the third row contains the values of IF1_ATTEN. Table 8 is an example of the data and its format. Since the IF bandwidth is typically less than 40 MHz wide and centered at a fixed frequency, it is sufficient to perform the calibration at the center IF as attenuation variation is insignificant over its range.

Table 8. An example of IF attenuation calibration.

IF3_Attenuator 2	0.973	1.927	2.948	3.912	.	.	.	.	28.630	29.634

IF3_Attenuator1	0.989	1.970	2.998	3.989					28.890	29.874
IF1_ Attenuator	0.995	2.028	3.038	4.023	.	.	.	.	28.868	29.854

RF calibration (0x9F8). This is a 33x50 float matrix. Table 9 is an example of the data and format for the RF calibration data. RF calibration contains data for preamplifier gain, gain with zero RF and IF attenuation, and attenuation values of the RF attenuator for fifty frequency points that span the operational frequency range of the SC5304A. Data is read in concatenation of rows. For example, all the frequency values are read in first, then the preamplifier gain values, followed by the zero attenuation-setting gain, etc. There are a total of 1650 values read that must be read from the EEPROM to form the full set of calibration.

Table 9. Example of the RF calibration data and its format.

Frequency (MHz)	3	5	10	.	500	.	.	.	3875	3900
Preamp Gain	20.564	20.643	20.456	.	20.003	.	.	.	19.654	19.231
Gain	33.223	33.423	33.213	.	33.102	.	.	.	29.980	29.450
RF_ATTEN 1 dB	0.988	0.955	1.093	.	0.973	.	.	.	1.008	0.995
RF _ATTEN 2 dB	1.921	2.001	2.045	.	2.056	.	.	.	1.932	2.051
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
RF_ATTEN 30 dB	29.645	29.854	30.065	.	29.588	.	.	.	29.260	29.572

Frequency Correction

On power-up, The SC5304A automatically applies the calibration value to the on-board reference DAC that controls the TCXO, which is the primary frequency reference of the device. The user may choose to reprogram the DAC with the 16 bit code obtained from the EEPROM at starting address 0x54, or with another value, by writing the REFERENCE_DAC register (0x17).

Gain Correction

The SC5304A has seven dynamic variables that affect its gain, namely, pre-amplifier state (on/off), IF attenuator settings, RF attenuator setting, filter path, inversion gain, input frequency, and temperature. Correction of gain needs to take into account five main factors. As noted in the "**EEPROM Data Content**" sub-section, the pre-amplifier gain, through gain (no attenuation, no pre-amplification), RF attenuation, and gain-over-temperature variation are calibrated over the span of the SC5304A frequency range. These are the frequency dependent parameters that are combined with the IF attenuation to make the total gain calculation.

Let us start by writing the gain equation with no dependence on temperature or frequency, and with the pre-amplifier turned on. We get the following equation.

$$G_{dev} = G_{preamp} + G + \Delta G_{fil} + \Delta G_{inv} - A_{rf} - A_{if} \qquad \text{Equation 1}$$

G_{dev} is the total gain of the device, G_{preamp} is the gain of the pre-amplifier, G is the through gain, A_{rf} is the attenuation of the RF attenuator, ΔG_{fil} is the gain change of IF3_FILTER1 path, ΔG_{inv} is the gain change in spectral inversion mode, and A_{if} is the attenuation of the the IF attenuators. If the preamplifier is off, no inversion, default filter path, and no attenuation applied then $G_{dev} = G$. Writing Equation 1 with dependency on temperature we add on the temperature dependent gain factor $\Delta G(T)$ and obtain the following:

$$G_{dev}(T) = G_{preamp}(T_0) + G(T_0) + \Delta G_{fil}(T_0) + \Delta G_{inv}(T_0) - A_{rf}(T_0) - A_{if}(T_0) + \Delta G(T) \qquad \text{Equation 2}$$

where T is the temperature of the device and T_0 is the fixed temperature at which calibration was performed. The section "**Reading Temperature Data**" provides information on how unsigned raw temperature data is converted to Celsius, a floating point type. Taking the frequency dependence of the measured parameters into consideration, Equation 2 may be rewritten as

$$G_{dev}(f,T) = G_{preamp}(f,T_0) + G(f,T_0) - A_{rf}(f,T_0) + \Delta G(f,T) + \Delta G_{fil}(T_0) + \Delta G_{inv}(T_0) - A_{if}(T_0) \qquad \text{Equation 3}$$

Note that the IF attenuation values, $A_{if}(T_0)$ do not need to be frequency dependent as discussed earlier. Using the IF attenuator calibration is as simple as substituting the intended value with the calibrated value. From Table 8, one would use 29.854 dB for an intended 30 dB attenuation. The other 2 variables, $\Delta G_{fil}(T_0)$ and $\Delta G_{inv}(T_0)$, are also frequency independent as they are only referred to at the center of the IF band, and their values are simply summed in the total gain equation. Only those parameters that depend on frequency and/or temperature are treated below.

To obtain calibrated gain values from the parameters that are a function of frequency, interpolation is required to provide the best estimated values. A natural cubic Spline interpolation is suggested for $G_{preamp}(f,T_0)$, $G(f,T_0)$, and $A_{rf}(f,T_0)$. The important input parameters for a cubic spline interpolation are the 2 arrays $[X]$ and $[Y]$, and an arbitrary point x. The output of the interpolation is some interpolated value y based on the inputs. $[X]$ is the set of independent values, $[Y]$ is the set of $[X]$ dependent values, and x is an arbitrary independent value to obtain the interpolated value y. For example, Table 10 lists the input and output parameters to obtain the gain $G(f = 1000\ MHz, T_0)$.

Table 10. Parameters to a Spline interpolation.

Frequency (MHz); $[X]$	3	5	19	…	…	950	1050	…	3875	3900
Measured Gain; $[Y]$	33.223	33.423	33.213	…	…	32.652	32.482	…	29.980	29.450
$f; x$	1000									
$G(f,T_0); y$	32.453									

From experience, having a large [X] and [Y] array of points does not necessarily provide the best interpolated value due the nature of trying to fit a function over many points and over many octaves of frequency. Better results are obtained from a set of localized calibrated points around the point of interest. The function **sc5304a_CalcGain** uses six localized [X] points to compute the interpolated point. Using localized points, the example on Table 10 is re-tabulated in Table 11. Similarly, frequency dependent preamplifier gain and RF attenuation may be derived.

Table 11. Localized parameters to a Spline interpolation.

Frequency (MHz); $[X]$	850	900	950	1050	1100	1150
Gain; $[Y]$	32.681	32.673	32.652	32.482	32.419	32.418
$f; x$	1000					
$G(f, T_0); y$	32.532					

To find the change in gain with respect to change in temperature involves a couple of steps; first determine the array values of $[\Delta G(f_i, T)]$, where f_i is the frequency point at which a measurement was made, and then as a second step, use interpolation to determine the $\Delta G(f, T)$ at some frequency f. Again, natural cubic spline interpolation is recommended in the second step. Let us outline a method to determine $\Delta G(f_i, T)$ at frequency f_i; there are a total of 8 frequency points for this calibration.

The calibration values retrieved from the EEPROM are second order polynomial coefficients fitted to measured data. Writing the general form of the gain function using coefficients, we have:

$$G(f_i, T) = a_0(f_i) + a_1(f_i)T + a_2(f_i)T^2 \qquad \text{Equation 4}$$

Here $a_j(f_i)$ is the j^{th} order coefficient measured some frequency f_i. The gain deviation at temperature T from the gain measured at the calibration temperature T_0 can be written as

$$\begin{aligned} \Delta G(f_i, T) &= G(f_i, T) - G(f_i, T_0) \\ &= a_1(f_i)[T - T_0] + a_2(f_i)\left[T^2 - {T_0}^2\right] \end{aligned} \qquad \text{Equation 5}$$

Using equation 5 and the temperature coefficients of **Table 6**, we obtain the following:

Table 12. Calculated gain changes at the measured frequency points

Frequency (MHz) (f_i)	50.0	250	500	1000	1500	2500	2800	3800
$\Delta G(f_i, T = 45^O)$ dB	-0.340	-0.343	-0.351	-0.348	-0.354	-0.351	-0.355	-0.357

After determining the set of gain deviations at some temperature T, we apply spline interpolation to the set of $\Delta G(f_i, T)$ values to obtain $\Delta G(f, T)$, change in gain with respect to both temperature and frequency. Using Table 12 and the convention developed here, the Spline parameters are $[X] = [f_i]$, $[Y] = [\Delta G(f_i, T)]$, $x = f$, and $y = \Delta G(f, T)$.

IF Response Correction

The gain correction procedure outlined above only applies to a signal that is centered in the 70 MHz IF band. The device's Fine-Tune mode (1 Hz) is able to place any RF signal at the center of the IF, so for narrow bandwidth signals (typically less than a MHz), applying the center IF gain correction and assuming no deviation from linear phase is sufficient. However, for a large bandwidth signal that spans several MHz, it is important to apply gain and phase correction to the offset frequencies; those that are offset from the center IF. Although SignalCore performs calibration of the amplitude and phase over the

bandwidth of the IF filters (available on the device calibration EEPROM), it is recommended that the user perform in-situ system equalization for digital broadband applications for improved performance.

Measured IF gain and phase error response is available for both filter paths; the user simply needs to properly select the path of interest. The measurement is made using a vector network analyzer in the frequency domain, covered by fifty-one evenly spaced frequency points. The amplitude (gain) error values are measured with respect to the center frequency and are given in decibels, while the phase error values are in radians. The phase errors are deviations from linear phase. Each set of calibrated points consists of a 3x51 floating point array (see Table 7 as an example). There are several ways to apply the frequency domain calibration:

1. Determine a fitted polynomial function for the amplitude error (gain) and multiply this function with the uncorrected amplitude spectrum. Add the two values if dealing in decibels. Additionally, determine a fitted polynomial function for the phase error and add values derived from this function with the uncorrected phase. To derive, let $X(e^{i\omega})$ be the measured uncorrected value, $E(e^{j\omega})$ be the fitted polynomial to the calibrated error values, and $Y(e^{j\omega})$ be the corrected measured value. Also let $Arg[\cdot]$ denote the principle value of the phase of the above terms, and we can relate all the terms as

$$Y(e^{j\omega}) = E(e^{j\omega})X(e^{j\omega})$$
$$|Y(e^{j\omega})|e^{jArg[Y(e^{j\omega})} = |E(e^{j\omega})||X(e^{j\omega})|e^{j(Arg[Y(e^{j\omega})]+Arg[Y(e^{j\omega})])}$$

   From the above equation, we see that the magnitude terms are multiplied and the phase terms added. In the discrete sense (digitized) for every frequency value, ω_i, we apply the above equation to correct for the non-ideality of the IF filter.

2. The other method finds the magnitude and error points through interpolation methods such as Spline, then multiplying the error magnitude with the uncorrected magnitude and adding the error and uncorrected phases. This is similar to method 1, but instead of using a fitted function to obtain the error values, interpolation is used. Interpolation is generally a slower process. This is the method implemented in the library function **sc5304a_CalcIfResponseCorrection**.

SOFTWARE API LIBRARY FUNCTIONS

SignalCore's philosophy is to provide products to our customers whose lower hardware functions are easily accessible. For experienced users who wish to use direct, low-level control of frequency and gain settings, having the ability to access the registers directly is a necessity. However, others may wish for simpler product integration using higher level function libraries and not having to program device registers directly. The functions provided in the SC5304A API dynamic-linked, shared, or LabVIEW libraries are:

- **sc5304a_SearchDevices**
- **sc5304a_OpenDevice**
- **sc5304a_CloseDevice**
- **sc5304a_RegWrite**
- **sc5304a_RegRead**
- **sc5304a_InitDevice**
- **sc5304a_SetStandby**
- **sc5304a_SetFrequency**
- **sc5304a_SetPreamp**
- **sc5304a_SetSignalChain**
- **sc5304a_SetSynthesizerMode**
- **sc5304a_SetIfFilterPath**
- **sc5304a_SetReferenceClock**
- **sc5304a_SetReferenceDac**
- **sc5304a_SetIfInversion**
- **sc5304a_WriteUserEeprom**
- **sc5304a_SetSignalPhase**
- **sc5304a_GetDeviceStatus**
- **sc5304a_GetTemperature**
- **sc5304a_ReadCalEeprom**
- **sc5304a_ReadUserEeprom**
- **sc5304a_ReadUserEepromBulk**
- **sc5304a_GetRawCalData**
- **sc5304a_GetCalData**
- **sc5304a_CalcAutoAttenuation**
- **sc5304a_CalcGain**
- **sc5304a_CalcIfResponseCorrection**
- **sc5304a_ConvertRawCalData**
- **sc5304a_ConvertRawTempData**
- **sc5304a_spline**
- **sc5304a_splineInterp**

Each of these functions is described in more detail on the following pages. To program in C/C++, SignalCore defines the following constants and types which are contained in the C header file, **sc5304a.h.** These constants and types are useful not only as an include file for developing applications using the sc5304a libraries, but also for writing device drivers independent of those provided by SignalCore.

Constants Definitions

```
/* 2-D parameters for storing calibration data */
#define RFCALPARAM                  33  //rows of caldata
#define RFCALFREQ                   50  // frequency points
#define IFATTENUATOR                3 // total number of IF attenuators
#define IFATTENCALVALUE             30  // attenuation steps 1-30 dB
#define IFRESPONSEPARAM             3  // freq,amp,phase
#define IFRESPONSEFREQ              51  // freq points over the bandwidth
#define TEMPCOPARAM                 3   // freq, coeff(1) , coeff(2)
#define TEMPCOFREQ                  8   // freq points

/* Attenuator assignment */
#define IF3ATTENUATOR2              0
#define IF3ATTENUATOR1              1
#define RFATTENUATOR                2
#define IF1ATTENUATOR               3

#define CALEEPROMSIZE               15168
#define USEREEPROMSIZE              16384

/* Tune mode parameters */
#define FASTTUNEENABLE              1
#define DISABLEDFINEMODE            0 //1 MHz tuning steps, PLL implementation
#define PLLFINEMONDE                1 //25 KHz tuning steps, PLL implementation
#define DDSFINEMODE                 2 //1 Hz tuning steps, DDS implementation

/* ERROR Set */
#define SUCCESS                     0
#define DEVICEERROR                 -1
#define TRANSFERERERROR             -2
#define INPUTNULL                   -3
#define COMMERROR                   -4
#define INPUTNOTALLOC               -5
#define EEPROMOUTBOUNDS             -6
#define INVALIDARGUMENT             -7
#define INPUTOUTRANGE               -8
#define NOREFWHENLOCK               -9
#define NORESOURCEFOUND             -10
#define INVALIDCOMMAND              -11
```

Type Definitions

```
typedef         unsigned char   bool;

typedef struct deviceAttribute_t
{
        unsigned int productSerialNumber;
        unsigned int rfModuleSerialNumber;
        float firmwareRevision;
        float loHardwareRevision;
        float scHardwareRevision;
        unsigned int *calDate; //size of 4 year,month,day,hour
        unsigned int *manDate; //size of 4 year,month,day,hour
}       deviceAttribute_t;

typedef struct calibrationData_t
{
        float **rfCal;     // RF gain calibration
        float **ifAttenCal; // IF attenuators calibration
        float **ifFil0ResponseCal; // IF filter 0 response calibration
        float **ifFil1ResponseCal; // IF filter 1 response calibration
        float **tempCoeff; // temperature coefficients
        float rfCalTemp; // temperature T0 at which calibration was done
        float ifFilter1GainError; // Gain error when switched to filter 1 path
        float ifFilter0Bw; // filter 0 BW in MHz
        float ifFilter1Bw; // filter 1 BW in MHz
        float invertGainError;  // gain error when spectral inversion enabled
        unsigned int tcxoDac;  // The TCXO dac value at T0
}       calibrationData_t;

typedef struct attenuator_t
{
        unsigned int if3Atten2Value;
        unsigned int if3Atten1Value;
        unsigned int rfAttenValue;
        unsigned int if1AttenValue;
}       attenuator_t;

typedef struct deviceStatus_t
{
        bool tcxoPllLock;
        bool vcxoPllLock;
```

```
        bool lo1Pll3Lock;
        bool lo2PllLock;
        bool lo3PllLock;
        bool lo1Pll1Lock;
        bool lo1Pll2Lock;
        bool extRefDetected;
        bool refClkOutEnable;
        bool extRefLockEnable;
        bool ifBandSelect;
        bool preampEnable;
        bool standbyEnable;
}       deviceStatus_t;

typedef struct ifResponseCorrect_t
{
        float ampCorrect;
        float phaseCorrect;
}       ifResponseCorrect_t;
```

Function Definitions and Usage

The functions listed below are found in the **sc5304a.dll** dynamic linked library for Windows operating systems or in the **libsc5304a.so** shared library for the Linux operating system. These functions are also provided in the LabVIEW library; **sc5304a.llb**. The LabVIEW functions contain context help (Ctrl-H) to provide further clarification of each function.

Function: **sc5304a_SearchDevices**

Definition: **int sc5304a_SearchDevices(char **serialNumberList)**

Return: The number of devices found

Output: char **serialNumberList (pointer list to serial numbers)

Description: **sc5304a_SearchDevices** searches for SignalCore SC5304A devices connected to the host computer and returns an array containing their serial numbers. The user can use this information to open the device(s) with their serial numbers. See **sc5304a_OpenDevice** function for information on how to open a device.

Function: **sc5304a_OpenDevice**

Definition: **deviceHande sc5304a_OpenDevice(char *devSerialNum)**

Return: device handle to the opened device

Output: char *devSerialNum (char pointer to a serial number, serial numbers are not limited to numerical values)

Description: **sc5304a_OpenDevice** opens the device and turns on the front panel access LED if successful. This function returns a handle of USB type to the device for other function calls.

Function: **sc5304a_CloseDevice**

Definition: **int sc5304a_CloseDevice(deviceHandle *devHandle)**

Return: The status of the function

Input: deviceHandle *devHandle (handle to the device to be closed)

Description: **sc5304a_CloseDevice** closes the device associated with the device handle and turns off the access LED of the front panel if it is successful.

Example:

To exercise the functions that open and and close the USB device:

```
// Declaring
char **deviceList;
deviceHandle    devHandle;
int devicesFound;
int i;
// Allocate memory
deviceList = (char**)malloc(sizeof(char*)*MAXDEVICES); // MAXDEVICES=50 serial numbers to search
for (i=0;i<MAXDEVICES; i++)
        deviceList[i] = (char*)malloc(sizeof(char)*SCI_SN_LENGTH); // SCI SN has 8 char

devicesFound = sc5304a_SearchDevices(deviceList);
printf("There are %d SignalCore USB devices found. \n", devicesFound);

i=0;
while ( i < numOfDevices)
        {
                printf(" Device %d has Serial Number: %s \n", i+1, deviceList[i]);
                i++;
        }

devHandle = sc5304a_OpenDevice(deviceList[0]); // get a handle to the first device on the List

// Free memory
for(i = 0; i<MAXDEVICES;i++) free(deviceList[i]);
free(deviceList);
//
// Do something with the device
//
//Close the device
int status = sc5304a_CloseDevice(devHandle);
```

Function: **sc5304a_RegWrite**

Definition: **int sc5304a_RegWrite(deviceHandle *devHandle,**
unsigned char commandByte,
unsigned int instructWord)

Return: The status of the function

Input: deviceHandle *devHandle (handle to the opened device)
unsigned char commandByte (the address byte of the register to write to)
unsigned int instructWord (the data for the register)

Description: **sc5304a_RegWrite** writes the instructWord data to the register specified by the commandByte. See the register map in Table 2 for more information. This function should rarely be used.

Example: To set the RF attenuator value to 10 dB:

```
int status = sc5304a_RegWrite(devHandle, 0x11, 0x020A);
```

Function: **sc5304a_RegRead**

Definition: **int sc5304a_RegRead(deviceHandle *devHandle,**
unsigned char commandByte,
unsigned int instructWord,
unsigned int *receivedWord)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
unsigned char commandByte (the address byte of the register to write to)
Unsigned int instructWord (the data for the register)
Unsigned int *receivedWord (data to be received)

Description: **sc5304a_RegRead** reads the data requested by the instructWord data to the register specified by the commandByte. See the register map in Table 3 for more information.

Example: To read the status of the device:

```
unsigned int deviceStatus;

int status = sc5304a_RegRead(devHandle,0x18,0x00,&deviceStatus);
```

Function: **sc5304a_SetStandby**

Definition: **int sc5304a_SetStandby(**
deviceHandle *devHandle,
bool standbyStatus)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

Bool standbyStatus (set to true (1) to set device in standby mode)

Description: **sc5304a_SetStandby** puts the device in standby mode where the power to the analog circuits is disabled, conserving power.

Function: **sc5304a_SetFrequency**

Definition: **int sc5304a_ SetFrequency(deviceHandle *devHandle, unsigned int frequency)**

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

unsigned int frequency (frequency in Hz)

Description: **sc5304a_SetFrequency** sets the RF frequency.

Function: **sc5304a_SetAttenuator**

Definition: **int sc5304a_SetAttenuator(**
deviceHandle *devHandle,
unsigned int attenValue,
unsigned int attenuator)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

unsigned int attenValue (the value assigned to the attenuator)

unsigned int attenuator (the designated attenuator, see header file)

Description: **sc5304a_SetAttenuator** sets the value of the designated attenuator.

Function: **sc5304a_SetPreamp**

Definition: **int sc5304a_SetPreamp(deviceHandle *devHandle, bool preampStatus)**

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

Bool preampStatus (turn on/off the preamp)

Description: **sc5304a_SetPreamp** enables or disables the RF preamplifier.

Function: **sc5304a_SetSignalChain**

Definition: **int sc5304a_SetSignalChain (**
deviceHandle *devHandle,
attenuator_t *atten,
bool preampStatus)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

unsigned int atten (values to the attenuators)

bool preampStatus (enable/disable the preamp)

Description: **sc5304a_SetSignalChain** sets all the attenuators and the preamp state. This simplifies the programming flow when used with **sc_5304a_CalcAutoAttenuation**, which returns the attenuator_t structure.

Example: Define attenuator_t atten and use it in the function:

```
attenuator_t *atten;
bool preamp = 0;
atten = (attenuator_t*)malloc(sizeof(attenuator_t)); //casting may not be necessary
// set for rfLevel = 0 dBm, mixerLevel = -20, ifLevel = 0 dBm, Pream off
atten->if3Atten2Value = 8;
atten->if3Atten1Value = 2;
atten->rfAttenValue = 20;
atten->if1AttenValue = 0;

int status = sc5304a_SetSignalChain(devHandle, atten, preamp);
free(atten);
```

Function: **sc5304a_SetSynthesizerMode**

Definition: **int sc5304a_SetSynthesizerMode(**
deviceHandle *devHandle,
bool fastTuneEnable,
unsigned int fineTuneMode)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
bool fastTuneEnable (enable/disable faster frequency stepping)
unsigned int fineTuneMode (selection of 1 MHz, 25 kHz, 1 Hz step resolution)

Description: **sc5304a_SetSynthesizerMode** enables/disables fast tuning, and sets the step resolution of the downconverter.

Function: **sc5304a_ SetIfFilterPath**

Definition: **int sc5304a_SetIfFilterPath(deviceHandle *devHandle, bool ifFilterPath)**

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
Bool ifFilterPath (selection of IF filter 0 or filter 1)

Description: **sc5304a_**SetIfFilterPath selects the IF filter**.**

Function: **sc5304a_SetReferenceClock**

Definition: **int sc5304a_SetReferenceClock(**
deviceHandle *devHandle,

bool lockExtEnable,
bool RefOutEnable,
bool Clk100Enable)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
bool lockExtEnable (enables phase locking to an external source)
bool RefOutEnable (enables the clock to driven out the REF OUT port)
bool Clk100Enable (changes REF OUT between 10MHz to 100 MHz)

Description: **sc5304a_**SetReferenceClock configures the reference clock behavior of the device**.**

Function: **sc5304a_SetReferenceDac**

Definition: **int sc5304a_SetReferenceDac(**
deviceHandle *devHandle,
unsigned int dacValue)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
Unsigned int dacValue (16-bit value for the reference DAC)

Description: **sc5304a_SetReferenceDac** set the value of the DAC that tunes the internal reference TXCO. The user may choose to override the value stored in memory to improve frequency accuracy.

Function: **sc5304a_SetIfInversion**

Definition: **int sc5304a_SetIfInversion(**
deviceHandle *devHandle,
bool ifInvertEnable)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
Bool ifInvertEnable (enable spectral inversion)

Description: **sc5304a_SetIfInversion** enables the down-converted signal to be spectrally inverted with respect the RF input. This may be beneficial for some applications.

Function: **sc5304a_SetSignalPhase**

Definition: **int sc5304a_SetSignalPhase(**
deviceHandle *devHandle,
float phase)

Return: The status of the function.

Input: deviceHandle *devhandle (handle to the opened device)
float phase (phase in degrees, 0-360 deg, 0.1 resolution)

Description: **sc5304a_SetSignalPhase** increases the phase of the signal by the amount specified.

Function: **sc5304a_WriteUserEeprom**

Definition: **int sc5304a_WriteUserEeprom(**
deviceHandle *devHandle,
unsigned int memAdd,
unsigned char byteData)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
unsigned int memAdd (memory address to write to)
unsigned char byteData (byte to be written to the address)

Description: **sc5304a_WriteUserEeprom** writes one byte of data to the memory address specified.

Function: **sc5304a_GetDeviceStatus**

Definition: **int sc5304a_GetDeviceStatus(**
deviceHandle *devHandle,
deviceStatus_t *deviceStatus)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

Output: deviceStatus *deviceStatus (outputs the status of the device such as PLL lock)

Description: **sc5304a_GetDeviceStatus** retrieves the status of the device such as LO phase-lock status, and current device settings.

Example: Code showing how to use this function:

```
deviceStatus_t *devStatus;
devStatus = (deviceStatus_t*)malloc(sizeof(deviceStatus_t));

int status = sc5304a_GetDeviceStatus(devHandle, devStatus);

if(devStatus->vcxoPllLock)
printf("The 100 MHz is phase-locked \n");
else
printf("The 100 MHz is not phase-locked \n");

free(deviceStatus);
```

Function: **sc5304a_GetTemperature**

Definition: **int sc5304a_GetTemperature (**
deviceHandle *devHandle,
float *temperature)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

Output: float *temperature (temperature in degrees C)

Description: **sc5304a_GetTemperature** retrieves the internal temperature of the device.

Function: **sc5304a_ReadCalEeprom**

Definition: **int sc5304a_ReadCalEeprom(**
deviceHandle *devHandle,
unsigned int memAdd,
unsigned char *byteData)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

unsigned int memAdd (EEPROM memory address)

Output: unsigned char *byteData (the read byte data)

Description: **sc5304a_ReadCalEeprom** reads back a byte from the memory address of the calibration EEPROM.

Function: **sc5304a_ReadUserEeprom**

Definition: **int sc5304a_ReadUserEeprom(**
deviceHandle *devHandle,
unsigned int memAdd,
unsigned char *byteData)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)

unsigned int memAdd (EEPROM memory address)

Output: unsigned char *byteData (the read byte data)

Description: **sc5304a_ReadUserEeprom** reads back a byte from the memory address of the user EEPROM.

Function: **sc5304a_ReadUserEepromBulk**

Definition: **int sc5304a_ReadUserEepromBulk(**
deviceHandle *devHandle,
unsigned int startMemAdd,
unsigned char *byteDataArray)

Return: The status of the function

Input: deviceHandle *devhandle (handle to the opened device)
unsigned int memAdd (EEPROM start memory address)
Output: unsigned char *byteData (the read 64 bytes of data)
Description: **sc5304a_ReadUserEepromBulk** reads back 64 bytes beginning at the start memory address of the user EEPROM.
Example: Code to read back 512 bytes of data starting at address 1024 into eepromData:

```
unsigned char eepromData = (unsigned char*)malloc(512);
unsigned char *bufferIn = (unsigned char*)malloc(64);
int i = 0;
int bufferCount = 0;
unsigned int add = 1024;

while(bufferCount < 8){
        int status = sc5304a_ReadUserEepromBulk(devHandle, add+bufferCount*64, bufferIn);
        for (i = 0; i < 64; i++) eepromData[i + bufferCount*64] = bufferIn[i];
        bufferCount ++;
}

free(bufferIn);
```

Function: **sc5304a_GetRawCalData**
Definition: **int sc5304a_GetRawCalData(**
deviceHandle *devHandle,
unsigned char *rawCalDataArray)
Return: The status of the function
Input: deviceHandle *devhandle (handle to the opened device)
Output: unsigned char *rawCalDataArray (the entire calibration EEPROM contents)
Description: **sc5304a_GetRawCalData** reads the entire calibration EEPROM into the rawCalDataArry. The array must be allocated for at least 15168 bytes.
Example: See code block example below for using **sc5304a_ConvertRawCalData**.

Function: **sc5304a_GetCalData**
Definition: **int sc5304a_GetCalData(**
deviceHandle *devHandle,
deviceAttribute_t *deviceAttributes,
calibrationData_t *calData)
Return: The status of the function
Input: deviceHandle *devhandle (handle to the opened device)
Output: deviceAttribute_t *deviceAttributes (device attributes)

calibrationData_t *calData (structured calibration data)

Description: **sc5304a_GetCalData** returns the device attributes such as serial number, calibration date, and also structured calibration data used for gain calculation and correction.

Example: See code block example for **sc5304a_ConvertRawCalData**.

Function: **sc5304a_ConvertRawCalData**

Definition: **int sc5304a_ConvertRawCalData (**
unsigned char *rawCalData,
deviceAttribute_t *deviceAttributes,
calibrationData_t *calData)

Input: unsigned char *rawCalData (entire cal EEPROM data in raw byte format)

Output: calibrationData_t *calibrationData (current calibration data)

deviceAttribute_t *deviceAttributes (device attributes)

Description: **sc5304a_ConvertRawCalData** organizes/decodes the entire 15168 bytes of raw calibration data read from the EEPROM and returns two data formats - deviceAttribute_t and calibrationData_t. The array rawCalData must contain valid calibration data, obtained by reading the EEPROM, and rawCalData, calData and deviceAttributes must have memory allocated.

Example: Allocating memory for the input and output parameters in C, and calling the function. Similarly, allocated memory must be de-allocated when no longer used or when the program quits.

```
//Declaring
calibrationData_t* calData;
deviceAttribute_t* devAttr;
unsigned char *rawCal;

//allocate memory for raw calibration
rawCal = (unsigned char*)malloc(sizeof(char)*CALEEPROMSIZE);

// Allocate memory; the user may use malloc() instead of calloc()
devAttr->calDate = (unsigned int*)calloc(4,sizeof(unsigned int));
devAttr->manDate = (unsigned int*)calloc(4,sizeof(unsigned int));

calData->rfCal = (float**)calloc(RFCALPARAM,sizeof(float*));
for(i = 0;i<RFCALPARAM;i++)
        calData->rfCal[i] = (float*)calloc(RFCALFREQ,sizeof(float));

calData->ifAttenCal = (float**)calloc(IFATTENUATOR,sizeof(float*));
for(i = 0;i<IFATTENUATOR;i++)
        calData->ifAttenCal[i] = (float*)calloc(IFATTENCALVALUE,sizeof(float));

calData->ifFil0ResponseCal = (float**)calloc(IFRESPONSEPARAM,sizeof(float*));
for(i = 0;i<IFRESPONSEPARAM;i++)
        calData->ifFil0ResponseCal[i] = (float*)calloc(IFRESPONSEFREQ,sizeof(float));

calData->ifFil1ResponseCal = (float**)calloc(IFRESPONSEPARAM,sizeof(float*));
for(i = 0;i<IFRESPONSEPARAM;i++)
        calData->ifFil1ResponseCal[i] = (float*)calloc(IFRESPONSEFREQ,sizeof(float));

calData->tempCoeff = (float**)calloc(TEMPCOPARAM,sizeof(float*));
for(i = 0;i<TEMPCOPARAM;i++)
        calData->tempCoeff[i] = (float*)calloc(TEMPCOFREQ,sizeof(float));

//read in raw calibration
int status = sc5304a_GetRawCal(devHandle, rawCal);
//Calling the function to structure the calibration data
status = sc5304a_convertRawCalData(rawCal, devAttr, calData);

//alternatively instead of calling the above 2 functions
status = sc5304a_GetCalData(devHandle, devAttr, calData);
```

Function: **sc5304a_CalcAutoAttenuation**

Definition: **int sc5304a_CalcAutoAttenuation(**
unsigned int frequency,
float inputRfLevel,
float inputMixerLevel,
bool preampEnable,
float nominalIfOutLevel,
bool if3Filter1Enable,
float temperature,
calibrationData_t *calData,
attenuator_t *attenuator);

Input:	unsigned int frequency	(input RF frequency in Hz)
	float inputRfLevel	(input RF level in dB)
	float inputMixerLevel	(input mixer level in dB)
	bool preampEnable	(preamplifier enabled)
	float nominalIfOutLevel	(nominal IF out level in dB)
	bool if3Filter1Enable	(Enable Filter 1 path in IF3)
	float temperature	(current device temperature)
	calibrationData_t * calData	(structured calibration data for the device)
Output:	attenuator_t *attenuator	(attenuation settings for RF, IF1, and final IF3 attenuators)

Description: **SC5304A_CalcAutoAttenuation** returns the set of attenuation settings for all the attenuators that will configure the SC5304A for best dynamic range operation based on user input parameters such as frequency, mixer level, etc. The values are calculated to maintain a good balance between the signal-to-noise dynamic range and the linearity dynamic range. The input parameters are based on those of traditional spectrum analyzers. The SC5304A downconverter is designed for best balanced dynamic range with -20 dBm power at the input mixer and 0 dBm nominal power at the IF output port. Each attenuator must have memory allocated before calling this function. *SC5304A_ConvertRawCalData* must be called or valid structure calibration data must be entered before calling this function.

Example: Code showing how to properly use this function:

```
//Declaring
Attenuator_t *atten;
float deviceTemp;

//call a function to return the device temperature
function_to_sc5304a_GetTemperature( &deviceTemp);

unsigned int rfFreq = 1000000000;  // 1.0 GHz
float rfLevel = 0;  // expecting a 0 dBm input signal
float mixerLevel = -20;  //set the mixer level requirement
bool preamp = 0;          // since input level is 0 dB, no need for a preamp
float ifLevel = 0; // to obtain a level clost to 0 dBm at the IF
bool filterPath = 0; // use the default filter path in the IF
float temp = deviceTemp;

//Calling the function
int status = sc5304a_CalcAutoAttenuation(rfFreq, rfLevel,mixerLevel,preamp,ifLevel,filterPath,temp,
atten);
```

Function: **sc5304a_CalcGain**

Definition: **int sc5304a_CalcGain(**
unsigned int frequency,
bool preampEnable,
bool ifInvertEnable,
bool if3Fil1Enable,
attenuator_t *atten,
float temperature,
calibrationData_t *calData,
float *conversionGain);

Input:	unsigned int frequency	(input RF frequency in Hz)
	bool preampEnable	(preamplifier enabled)
	bool ifInvertEnable	(enable IF spectral inversion)
	attenuator_t *atten	(attenuation settings)
	float temperature	(temperature value of the device in degrees Celsius)
	calibrationData_t *calData	(calibration data for the device)
Output:	float *conversionGain	(calculated calibrate conversion gain for current settings)

Description: **sc5304a_CalcGain** calculates the calibrated gain based on the current user settings.

Function: **sc5304a_CalcIfResponseCorrection**

Definition: **int sc5304a_CalcIfResponseCorrection (**
float *offsetFrequencies,
unsigned int nPoints,
bool ifFil1PathEnable,
calibrationData_t *calData,
ifResponseCorrect_t *correctedIfResponse)

Input: float *offsetFrequencies (floating point number 1-D array)
unsigned int nPoints (number of points in the 1-D array)
bool ifFil1PathEnable (IF filter 1 path enable)
calibrationData_t *calibrationData (calibration data for the device)

Output: float *correctedIfResponse

Description: **SC5304A_CalculateIfResponseCorrection** determines the IF correct response for the set of IF offset frequencies. These offset frequencies may be the frequency components of an FFT of the acquired data being offset from its center frequency. To obtain the offset frequencies, one can simply subtract the frequencies from the IF center frequency. For example, if a digitizer sampling at 100 MHz is used to digitize the 70 MHz IF signal with bandwidth of 3 MHz, the center of the digitized signal is 30 MHz +/- 1.5 MHz. After performing digital spectral inversion and performing an FFT, take the subset of frequency components from 28.5MHz to 31.5 MHz and subtract 30 MHz to obtain the offset frequencies of -1.5 MHz to 1.5 MHz. Use this set of offset frequencies to compute the gain and phase corrections to be applied to the original signal spectrum at 28.5 MHz to 31.5 MHz. This algorithm may not be sufficient for computation of broadband signals due to the lack of computation speed and correction accuracy. The calibration stored does not account for in-band phase and amplitude variations due to temperature and these variations may cause sufficient errors, especially in broadband digital signals. The user should apply in situ equalization to correct for the in-band amplitude and phase errors.

Function: **SC5304A_ConvertRawTempData**

Definition: **int sc5304a_ConvertRawTempData(**
unsigned int rawTempData,
float *temperature)

Input: unsigned int rawTempData (16-bit rawTempData stored in a 32-bit unsigned int)

Output: float *temperature (temperature value of the device in degrees Celsius)

Description: **SC5304A_ConvertRawTempData** converts the rawTempData into a floating point number.

Function: **sc5304a_spline**

Definition: **int sc5304a_spline(**
double *xArray,
double *yArray,
int nPoints,
double firstBoundary ,
double secondBoundary ,
double *yInterpolant)

Input: double *xArray (the set of independent values)
double *yArray (the set of x-dependent function values; size is the same as xArray)
int nPoints (the number of points in xArray)
double firstBoundary (the second derivative of the first point in the set)
double secondBoundary (the second derivative of the last point in the set)

Output: double *yInterpolant (the return set of interpolants)

Description: Returns the Spline interpolants of the input parameters.

Function: **sc5304a_splineInterp**

Definition: **int sc5304a_splineInterp(**
double *xArray,
double *yArray,
double *yInterpolant,
double nPoints,
double x,
double *interpolatedYValue)

Input: double *xArray (the set of independent values)
double *yArray (the set of x-dependent function values; size is the same as xArray)
double *yInterpolant (the returned interpolant from spline())
int nPoints (the numbers of points in xArray)
double x (the value at which interpolation is performed)

Output: double *interpolatedYValue (the corresponding interpolated value at x)

Description: Returns the Spline interpolated value.

CALIBRATION & MAINTENANCE

The SC5304A is factory calibrated and ships with a certificate of calibration. SignalCore strongly recommends that the SC5304A be returned for factory calibration every 12 months or whenever a problem is suspected. The specific calibration interval is left to the end user and is dependent upon the accuracy required for a particular application.

SC5304A calibration data is stored in the RF module (metal housing). Therefore, changing or replacing interface adapters will not affect unit calibration. However, SignalCore maintains a calibration data archive of all units shipped. Archiving this data is important should a customer need to reload calibration data into their device for any reason. SignalCore also uses the archived data for comparative analysis when units are returned for calibration.

Should any customer need to reload calibration data for their SC5304A, SignalCore offers free support through support@signalcore.com. SignalCore will provide a copy of the archived calibration data along with instructions on how to upload the file to the SC5304A.

The SC5304A requires no scheduled preventative maintenance other than maintaining clean, reliable connections to the device as mentioned in the “Getting Started” section of this manual. There are no serviceable parts or hardware adjustments that can be made by the end user.

SC5304A Accessories

Port Accessories

SignalCore offers between-series adapters for changing connectivity on all RF I/O ports to suit specific requirements.

SignalCore's RF coaxial terminations provide a resistive power termination wherever it is needed with minimum reflection. They are recommended to properly terminate unused ports to ensure the lowest-noise installation.

Order Number	Description
7109001-01	Adapter, Type N Male to SMA Female, to 18 GHz (VSWR 1.15 @ 18 GHz, Stainless Steel Body, RoHS Compliant)
7109002-01	Adapter, BNC Male to SMA Female, to 4 GHz (VSWR 1.2 @ 4 GHz, Nickel-plated Brass Body, RoHS Compliant)
7109003-01	Adapter, SMA Male to BNC Female, to 4 GHz (VSWR 1.2 @ 4 GHz, Nickel-plated Brass Body, RoHS Compliant)
7109004-01	Termination, Type N Male, to 18 GHz (VSWR 1.25 @ 18 GHz, 2 Watts Dissipating, 50 Ω Nickel-plated Brass Body, RoHS Compliant)
7109005-01	Termination, SMA Male, 50 Ω, 1 Watt (VSWR 1.2 @ 18 GHz, 1 Watt Dissipating, 50 Ω, Stainless Steel Body, RoHS Compliant)
7109006-01	Termination, BNC Male, 50 Ω, 2 Watt (VSWR 1.15 @ 3 GHz, 2 Watts Dissipating, 50 Ω, Nickel-plated Brass Body, RoHS Compliant)
7109012-01	Cable Assy, USB, Type A to Type B, 2 m, Black

APPENDIX A

SPECIFICATIONS

Definition of Terms

The following terms are used throughout this datasheet to define specific conditions:

Specification (spec) Defines guaranteed performance of a calibrated instrument under the following conditions:

- 3 hours storage at room temperature (standardized to 25 °C) followed by 30 minutes minimum warm-up operation
- Specified environmental conditions are met within the specified operating temperature range of 0 °C to 40 °C unless otherwise noted
- Recommended calibration intervals are used

Typical data (typ) When used with <, > or in a range, defines performance met by approximately 80% of all instruments manufactured. This data is not guaranteed, does not include measurement uncertainty, and is valid only at room temperature (standardized to 25 °C).

Nominal values (nom) Characterizes product performance by means of average performance of a representative value for the given parameter (e.g. nominal impedance). This data is not guaranteed and is valid only at room temperature (standardized to 25 °C).

Measured values (meas) Characterizes expected product performance by means of measurement results gained from individual samples.

Specifications are subject to change without notice. For the most recent product specifications, visit www.signalcore.com.

Spectral Specifications

RF input range [1] ..... 1 MHz to 3.9 GHz

IF output center frequency ..... 70 MHz

IF output polarity [2] ..... Non Inverted/Inverted

IF Bandwidth (3 dB)

Final IF filter bypassed ..... > 20 MHz

Final IF filter enabled (Standard)..... > 18 MHz

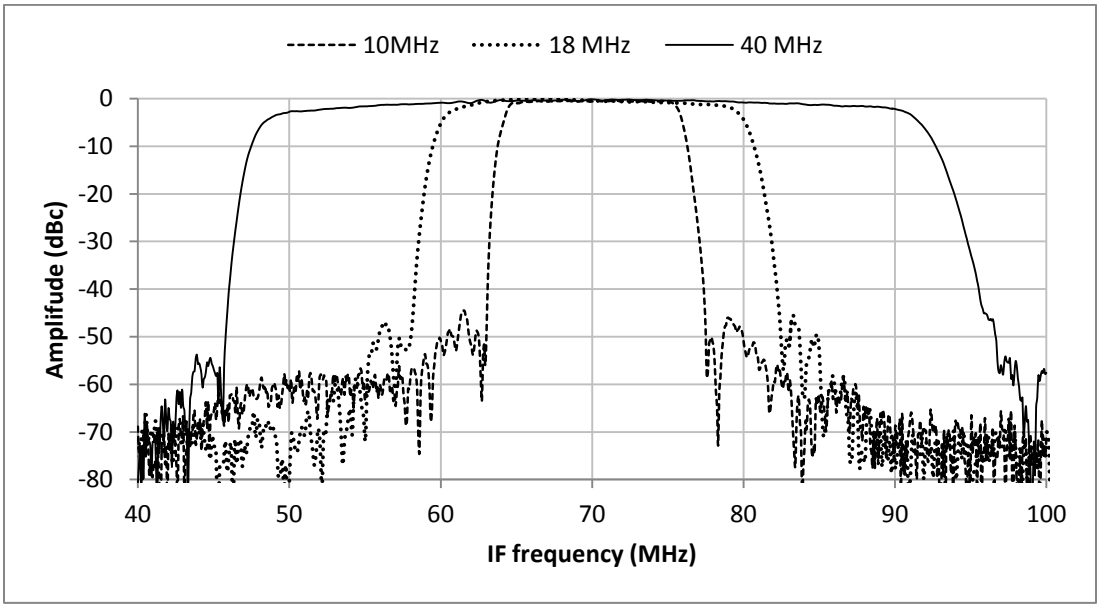


Figure 6. Typical output IF response of filter options. Maximum of two IF filter options are available. Standard product filter has 20 MHz bandwidth and bypass path.

(1) RF input below 1 MHz suffers from amplitude roll-off and calibration is not valid below this lower end frequency. In the frequency range below the specified IF bandwidth (< ~15 MHz) the first LO leakage appears inside the IF band. This LO leakage will appear as ~DC when the RF is converted to baseband in the final analysis. Furthermore, because the LO appears inside the IF band it will inter-modulate with the input RF signal to produce higher order in-band spurious signals that may degrade signal integrity. It is recommended to attenuate the RF signal before the mixer by applying RF attenuation or attenuate after the first mixer by applying IF1 attenuation. Suppressing the RF amplitude up front of the downconverter path will reduce the spurious signal levels.

(2) The IF output polarity refers to the conversion polarity of the downconverter. When the polarity is inverted the spectral content of at the output is inverted with respect to the input; this process is commonly known as “spectral inversion” or “spectral flipping”. The choice depends on the application. For digitizers that are sampling the IF in the even order Nyquist zones that naturally inverting spectra, having the IF polarity inverted will produced non-inverted baseband, and vise versa. However this is only a convenience in this application case because inverted spectrum, once digitized, can easily be re-inverted mathematically.

RF tuning

Resolution [3] ..... 1 Hz

Lock and settling times [4] ..... 1 ms

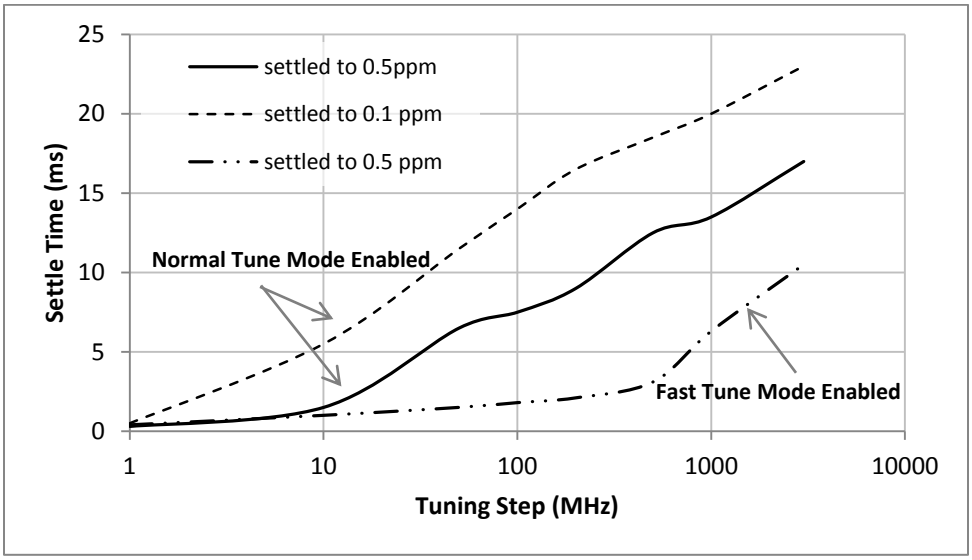


Figure 7. Typical frequency settle time vs tuning step with 3600 MHz being the final frequency

(3) To give the user flexibility, the DEVICE has 4 resolution modes; 2 coarse modes and 2 fine modes. The coarse modes using fractional N PLL allow 1 MHz and 50 kHz steps while the fine modes using PLL and DDS provide less than 1 Hz resolution. See the DEVICE user manual for further information.

(4) Lock and settled to < 1 ppm of final frequencies of >500 MHz and step size of <10 MHz. For final frequencies <500 MHz the settle time applies to accuracy with 500 Hz of the final frequency for a 10 MHz step. See figure X for example of other tuning step settling times. When fast-tune mode is enabled the noise damping capacitor across the main YIG tuning coil is disengaged, resulting in an increase of the rate of current flow through the coil and settle to a steady state quicker. Lock time begins when the full tuning word command is received by the device.

Frequency reference [5]

Technology ..... Temperature compensated crystal oscillator

Accuracy .....± [(aging x last adjustment time lapse) + temp stability + cal accuracy]

Initial calibration accuracy ..... ±0.05 ppm

Temperature stability [6]

20 ^{0}C to 30 ^{0}C ..... ±0.25 ppm

0 ^{0}C to 55 ^{0}C ..... ±1.0 ppm

Aging ..... ±1 ppm | First year @ 25 ^{0}C

Frequency accuracy [7] ..... ± (frequency reference accuracy * RF input frequency) Hz

(5) The frequency reference refers to the DEVICE internal 10 MHz TCXO time-base. Accuracy is in parts-per-million or ppm ($1x10^{-6}$)

(6) The customer must apply sufficient cooling to the device to keep the unit temperature as read from its internal temperature sensor within the range of 40 ^{O}C to 45 ^{O}C at an ambient temperature of 25 ^{O}C.

(7) This is the accuracy of the device for any given input RF signal.

Sideband phase noise (dBc/Hz) [8][9][11]

RF Frequency

Offset	100 MHz	1000 MHz	2000 MHz	3500 MHz
100 Hz	-88	-87	-85	-83
1 kHz	-100	-99	-98	-97
10 kHz	-108	-107	-106	-105
100 kHz	-120	-119	-118	-117
1 MHz	-143	-142	-142	-141
10 MHz	-152	-152	-150	-148

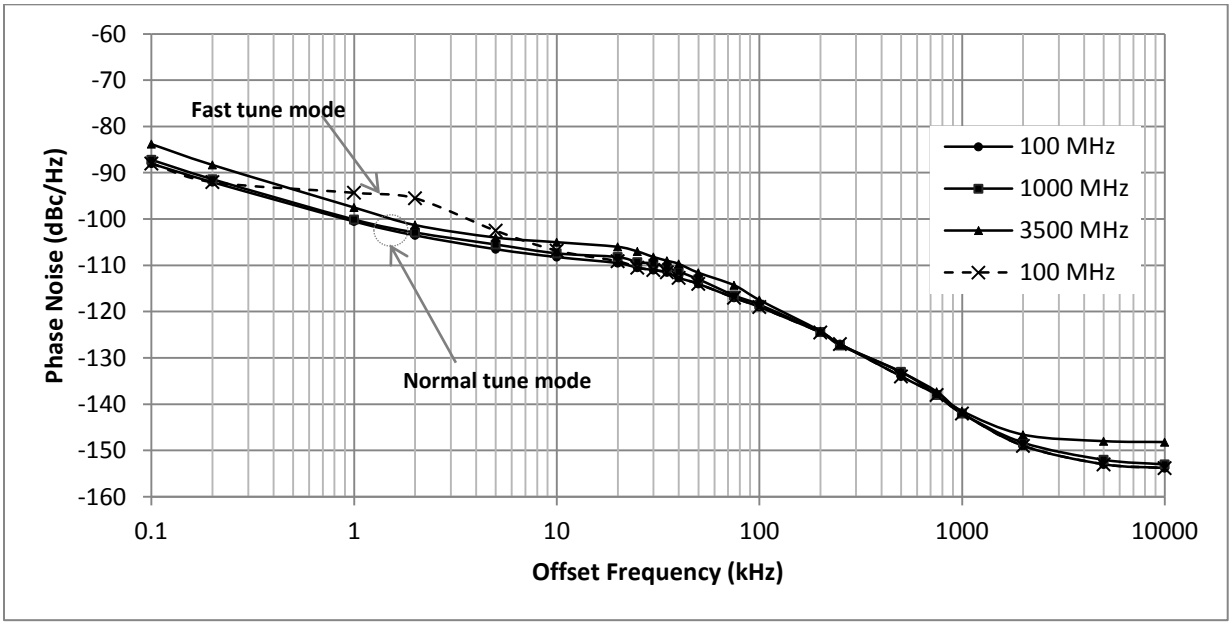


Figure 8. Typical measured sideband noise[9].

(8) Sideband phase noise as specified is based on measured sideband noise which includes both phase noise and amplitude noise contributions. Sideband noise is specified for the downconverter tune mode is set to NORMAL. See the user manual for further information how to set the device to NORMAL or FAST-TUNE modes.

(9) These results are obtained with input signal levels of 0 dBm at the mixer (no RF attenuation) and the output IF level was set to 3 dBm. The source is an ultra-low noise 100 MHz OCXO with noise floor of -176 dBc/Hz. The 1000 MHz and 3500 MHz signals were multiplied up from the same OCXO. The floor of the multiplied up 3500 MHz signal was about -143 dBc/Hz so a phase-locked YIG oscillator was used to complete the measurement for offset frequencies greater than 500 kHz. The YIG oscillator noise floor was better than -160 dBc/Hz. In FAST-TUNE mode the noise damping capacitor across the YIG tuning coil is disengaged and as a result the close-in phase noise degrades.

LO related sideband spurious signals (10)(11)

< 200 kHz ..... -75 dBc

> 200 kHz ..... -80 dBc

(10) Sideband spurious signals are results of the local oscillators in the DEVICE. Sources of sideband spurious signals in the synthesized local oscillators are primarily due to fractional-N spurious products in PLL, DDS noise sources, and inter-modulation between oscillators within the multiple-loop PLL synthesizers. Fractional-N and DDS spurious products affect spectral region below 200 kHz and inter-modulation products affect spectral regions out to a couple of MHz. SignalCore uses mathematical algorithms to properly select the synthesizer parameters used in the multiple-loop fractional-N PLL to ensure that typical sideband spurious products are better than the specifications.

(11) Specifications are valid for all modes of frequency tuning, whether is it PLL only mode or DDS driven mode. As the YIG oscillator is sensitive to magnetic fields, magnetic noise due to electrical fans, supply transformers, and other magnetic producing devices may induce sideband noise on the signals when they are place in close proximity. It is recommended that users should exercise good technical judgment when such devices needed; an example is mounting a cooling fan on the downconverter.

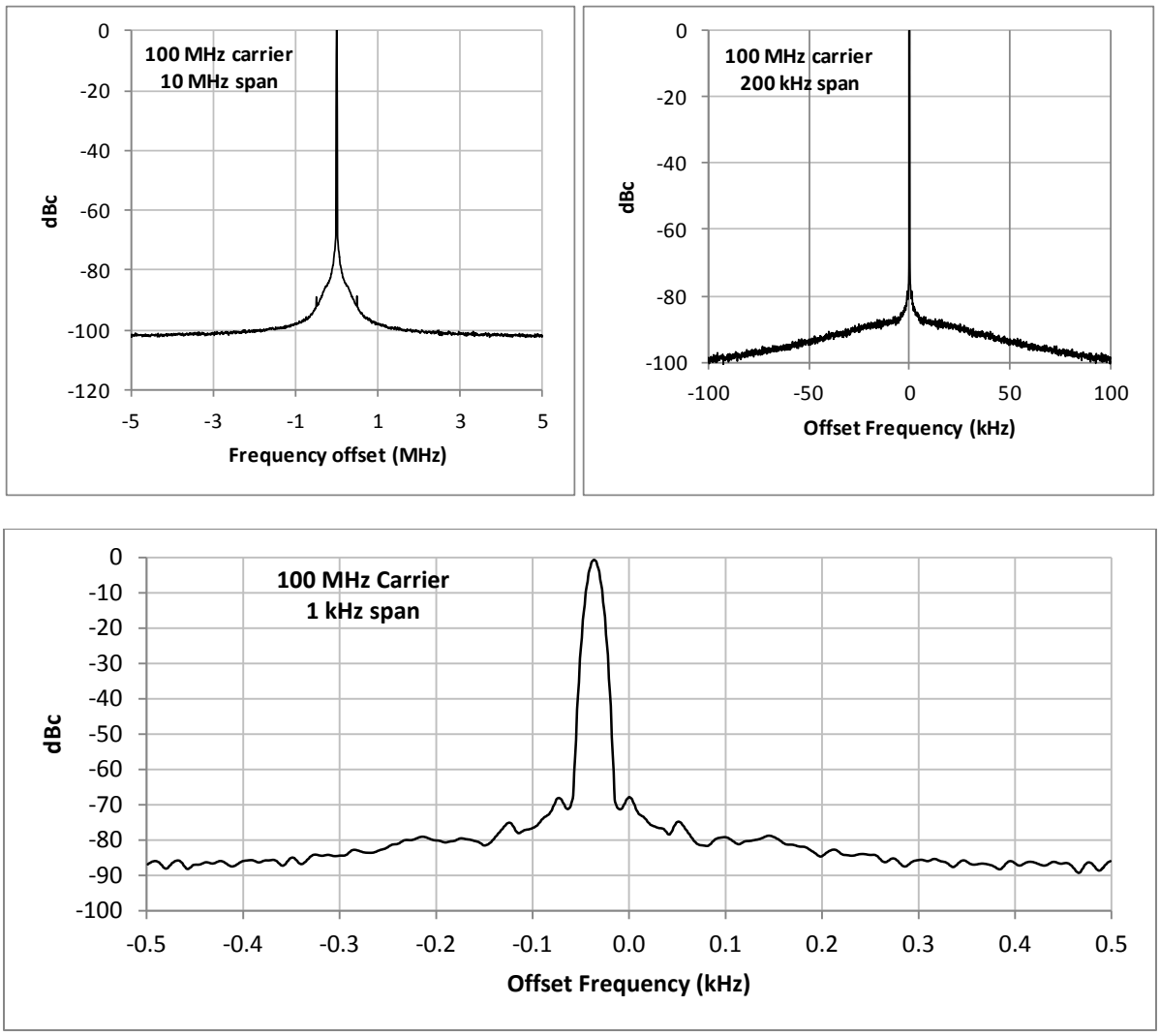


Figure 9. Plots show the raw spectral purity for a 100 MHz input RF signal (LO = 4.775 GHz). Note that the power supply noise of 60 Hz and its harmonics are in the noise.

Amplitude Specifications

Input Range

AC, preamp disabled ..... +27 dBm max
AC, preamp enabled ..... +23 dBm max
DC [12] ..... 0 V

Attenuation range

RF ..... 0 to 30 in 1 dB steps
IF [13] ..... 0 to 90 in 1 dB steps

Input voltage standing wave ratio (VSWR)

Preamp Off, 0 dB input attenuation

10 MHz to 2.4 GHz ..... < 1.5
2.5 GHz to 3.6 GHz ..... < 1.75

Preamp On, 0 dB input attenuation

10 MHz to 2.4 GHz ..... < 1.5
2.5 GHz to 3.6 GHz ..... < 1.9

Gain range (@ 1GHz) [14]

Minimum [15] ..... -60 dB typical
Maximum (Preamplifier disabled) [16] ..... 30 dB typical
Maximum (Preamplifier enabled) [16] ..... 50 dB typical

Preamplifier gain ..... 20 dB typical

(12) Large and fast DC transients could damage the input solid state devices. Slow ramp up of DC to 10 V is sustainable.

RF Amplitude accuracy (15 °C to 35 °C ambient)

RF gain response flatness (uncorrected) ..... 8 dB typ
RF gain flatness response (corrected) [17] ..... ±0.75 dB
Absolute gain accuracy (corrected) [17] ..... ±0.9 dB (±0.5 dB typ)

IF Amplitude accuracy (15 °C to 35 °C ambient)

IF in-band response flatness (uncorrected) ..... 3 dB typ
IF in-band response (corrected) [16] ..... ±0.5 dB

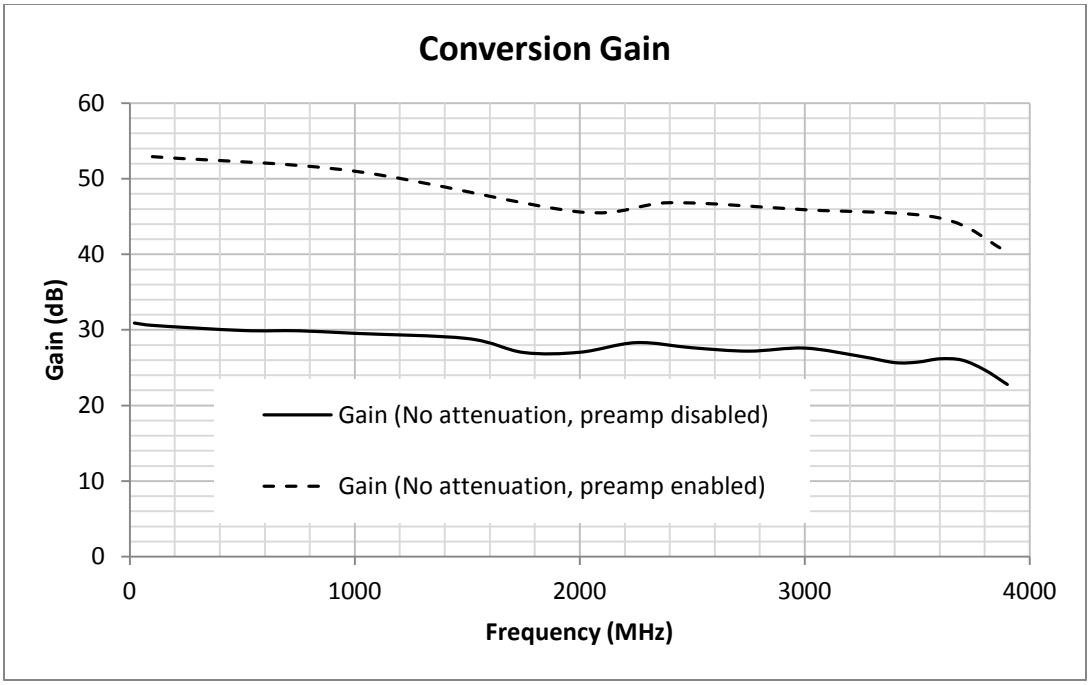


Figure 10. Typical RF conversion gain response @ 25 ^{O}C

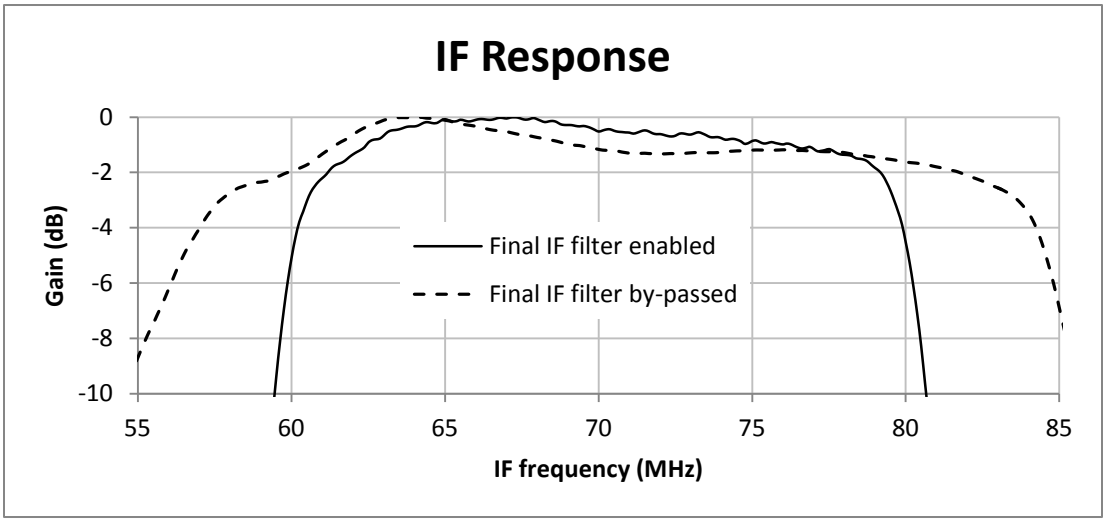


Figure 11. Typical IF amplitude response @ 25 ^{O}C

RF to IF group delay (80% of IF bandwidth)

IF Filter 0 path ..... 1 us typical

IF Filter bypassed .....100 ns typical

IF Phase Linearity (80% of IF bandwidth) (18)

IF Filter 0 path..... +/- 8 deg.

IF Filter bypassed .....+/- 8 deg

IF Phase Linearity deviation rate.....< 2 deg/MHz

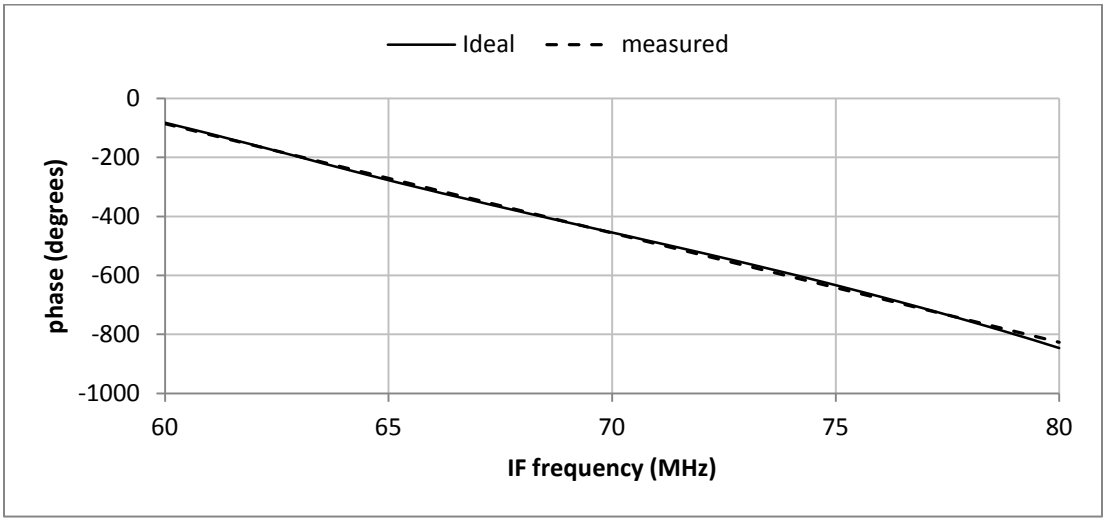


Figure 12. Phase deviation over 20 MHz

(13) There are 3 IF attenuators in total, each having 30 dB of attenuation. There are 2 attenuators in the final stage, and 1 attenuator in the first IF stage after the first mixer. How to effectively use them to optimize for performance is outlined in the DEVICE user manual

(14) These are typical gain specifications. The gain of the device is calibrated and stored in the device calibration EEPROM.

(15) Minimal gain is specified when all attenuators are set to their maximum values and the RF pre-amplifier is disabled.

(16) Maximum gain is specified when all the attenuators are set to 0 dB.

(17) Correction stored in the calibration EEPROM must be applied properly. Users are not obligated to use the calibration provided, they could devise their own method of calibration and correction should they choose to. User method of calibration and application may improve on the accuracies specified.

(18) For broadband signal operation it recommended that users apply in situ amplitude and phase equalization to the received signal to minimize amplitude and phase errors caused by the DEVICE. Phase deviation at offset frequencies from the center frequency of 70 MHz is stored in the calibration EEPROM. The calibration may be applied as a first order correction.

Dynamic Range Specifications

Spurious response (19)

Residual spurious signals (20) ..... < -100 dBm
LO related spurious signals (21) ..... < -80 dBc
Image rejection (22) ..... <-100 dBc
IF rejection (23) ..... <-115 dBc

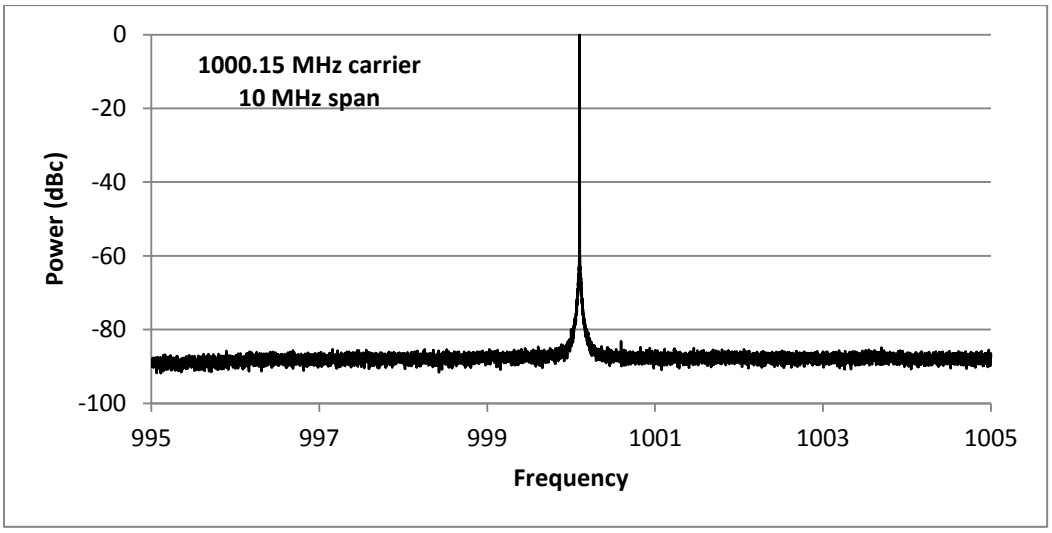


Figure 13. Spectrum showing low LO related spurious signals at for input signal 1000.15 MHz

(19) Spurious responses are unwanted signals appearing at the IF output. All spurious products are treated as if they originate at the input port of the downconverter, which is the common reference plane as the RF input.

(20) Residual spurious signals are observed and reference to the RF input of the downconverter when the RF input is terminated with a matched load. The RF and first IF (IF1) attenuators are set to 0 dB attenuation and the final IF attenuators were adjusted to obtain a downconverter gain to 20 dB. The preamplifier is disabled.

(21) LO related spurious signals are unwanted signals produced at the IF output due to inter-modulation of all the downconverter local oscillators. These spurious signals are measured relative to a RF signal present at the input. Specification provided here is when the downconverter is configured for -20 dBm at the mixer, and 0 dBm at the IF output, and a total gain of 20 dB.

(22) Image rejection is the DEVICE ability to reject an image signal of the RF frequency that would otherwise produce the same result as the desired RF signal. The image of the desired RF signal is calculated as: $RF_{image} = RF + 2IF_1$, where $IF_1 = 4.675\ GHz$.

(23) IF rejection is the DEVICE ability to reject RF signals at any of the IF frequencies while the downconverter is tuned elsewhere. Signal level at the mixer is -20 dBm and gain of 20 dB.

Input Noise (15 °C to 35 °C ambient) [24]

Preamplifier disabled [25]

	100 MHz	1000 MHz	3600 MHz
Noise floor (dBm/Hz)	-153	-152	-148
Noise figure (dB)	21	22	26

Preamplifier enabled [25]

	100 MHz	1000 MHz	3600 MHz
Noise floor (dBm/Hz)	-167	-166	-164
Noise figure (dB)	7	8	10

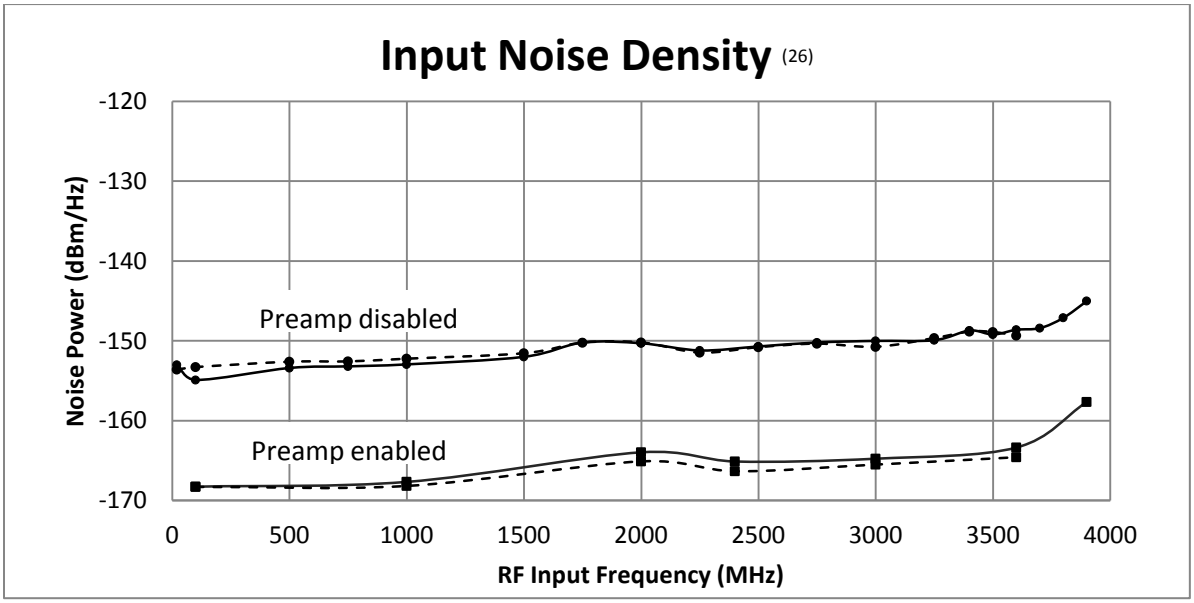


Figure 14. Measured noise density of the average of 2 lots.

(24) Noise (thermal) is referred to the input of the downconverter.

(25) The downconverter is configured with 0 dB RF attenuation, 0 dB IF1 attenuation, and IF attenuators set the gain to 20 dB. This setting is made to be consistent with the configuration for other specifications such as linearity and spurious responses so that the user may obtain a clearer picture of the performance of the downconverter. The RF input is terminated with a matched 50 Ω load.

(26) In spectrum analyzer and signal analyzer applications this is also commonly referred to as the Displayed Average Noise Level (DANL). This assumes that the digitizer used does not limit the performance of the downconverter.

Input third-order intermodulation (IIP3, dBm)

	100 MHz – 1 GHz	1 GHz – 2.5 GHz	2.5 GHz – 3.9 GHz
Preamplifier disabled (27)(29)	16 [17]	17.5 [20]	18.5 [20]
Preamplifier enabled (28)(29)	-5.0 [-2]	-3.0 [-1]	-2.0 [0]

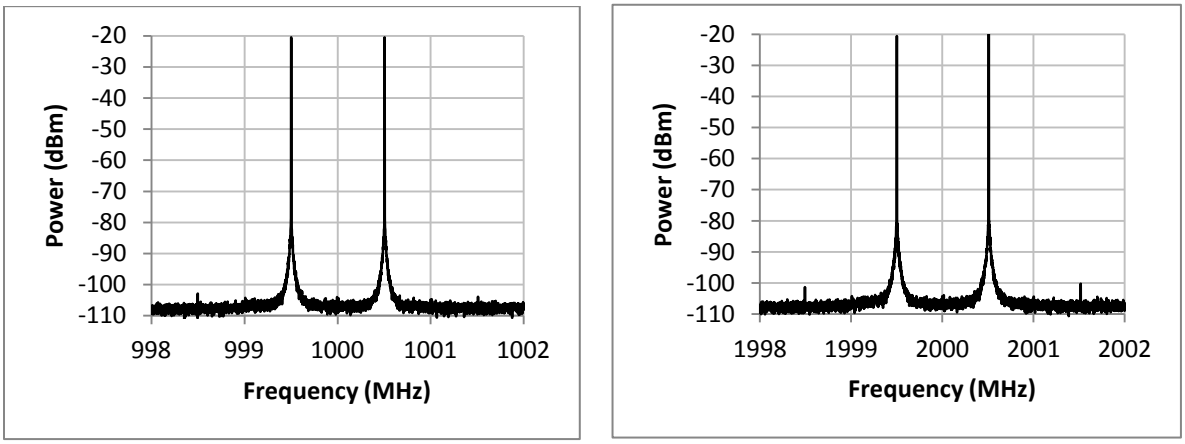


Figure 15. Plots show the typical IMD performance with two -20 dBm signals at the input, 0 dB RF attenuation, preamp disabled, and conversion gain of 20 dB

(27) Specifications are based on 0 dB RF attenuation, 0 dB IF1 attenuation, two -20 dBm tones with 1 MHz separation at the mixer, final IF attenuators set to maintain 0 dBm at the IF output.

(28) Specifications are based on 0 dB RF attenuation, 0 dB IF1 attenuation, two -30 dBm tones with 1 MHz separation at the mixer, final IF attenuators set to maintain 0 dBm at the IF output.

(29) These are in-band measurements and not out-of-band measurements. Out of band signal tones exist outside the IF filter bandwidth of the device, and thus may provide better IP3 measurements. However, using in-band signal tones provide better estimation of the device non linear effects on broadband signals.

Input second harmonic distortion (SHI, dBm)

Input second harmonic intercept point (dBm)	400 MHz	1000 MHz	1.8 GHz
Preamplifier disabled	62	62	58
Preamplifier enabled	32	33	30

Input compression point (dBm)

	100 MHz – 1 GHz	1 GHz – 2.5 GHz	2.5 GHz – 3.9 GHz
Preamplifier disabled	1	1.5	2
Preamplifier enabled	-23	-20	-19

Dynamic range

Measurement dynamic range [30] ..... > 185 dB

Instantaneous dynamic range [31] ..... > 150 dB

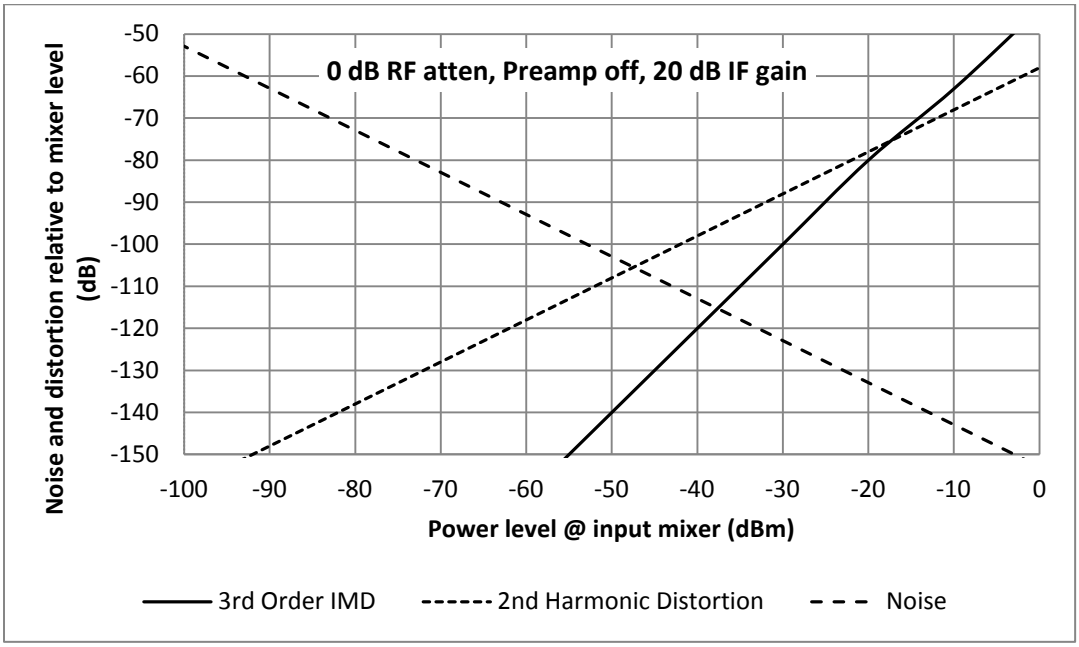


Figure 16. Instantaneous dynamic ranges plotted with preamplifier disabled for 1000 MHz measured data. Mixer level is at input level.

(30) Measurement dynamic range refers to the device SNR measurement capability using 2 or more configurations settings. For example, the user could set in sufficient RF attenuation to capture the high level signals and then turn on the preamplifier to measure low level noise.

(31) Instantaneous dynamic range refers to the instantaneous device SNR measurement using a single configuration setting. For example, the user could set the downconverter to receive a 0 dBm signal at the mixer, while at the same setting be able to measure the signal noise floor to -150 dB below its peak.

Reference Inputs and Outputs

Reference output specifications

Center frequency [32] ..... 10 MHz/100 MHz
Amplitude ..... 3 dBm typ
Waveform ..... Sine
Impedance ..... 50 Ω nominal
Coupling ..... AC
Connector Type ..... Female SMA
Frequency accuracy ..... See Frequency reference

Reference input specifications

Center frequency ..... 10 MHz
Amplitude ..... -10 dBm min/ +13 dBm max
Phase-lock range ..... ± 10 ppm (typ)
Impedance ..... 50 Ω nominal
Coupling ..... AC
Connector type ..... Female SMA

(32) The output reference frequency may be selected programmatically for 10 MHz or 100 MHz. The 100 MHz may be used to drive digitizer ADC directly.

Port Specifications

RF input

Input impedance ..... 50 Ω
Coupling ..... AC
Connector type ..... SMA female
LO leakage ..... <-120 dBm

IF output

Output impedance ..... 50 Ω
VSWR 1.6
Coupling ..... AC
Connector type ..... SMA female
Output amplitude ..... 20 dBm max

General Specifications

Environmental

Operating temperature [1] ..... 0 °C to +40 °C

Storage temperature [2] ..... -40 °C to +70 °C

Operating relative humidity [3] ..... 10% to 90%, non-condensing

Storage relative humidity [4] ..... 5% to 90%, non-condensing

Operating shock [5] ..... 30 g, half-sine pulse, 11 ms duration

Storage shock [6] ..... 50 g, half-sine pulse, 11 ms duration

Operating vibration [7] ..... 5 Hz to 500 Hz, 0.31 g_{rms}

Storage vibration [8] ..... 5 Hz to 500 Hz, 2.46 g_{rms}

Altitude ..... 2,000 m maximum (maintaining 25 °C maximum ambient temperature)

Physical

Dimensions (W x H x D, max envelope)..... 8.5” x 1.75” x 12.5”

Weight ..... 6.1 lbs

Power consumption ..... 34 W typical

Input voltage ..... ~90 – 264 VAC

Communication interface ..... USB

Safety ..... Designed to meet the requirements of:

IEC 61010-1, EN 61010-1, UL 61010-1, CSA 61010-1

Electromagnetic Compatibility (EMC) ..... Designed to meet the requirements of:

EN 61326-1 (IEC 61326-1): Class A emissions; Basic immunity 1, EN 55011 (CISPR 11) Group 1, Class A emissions, AS/NZS CISPR 11: Group 1, Class A emissions, FCC 47 CFR Part 15B: Class A emissions, ICES-001: Class A emissions

CE ..... Meets the requirements of:

2006/95/EC; Low-Voltage Directive (safety), 2004/108/EC; Electromagnetic Compatibility Directive (EMC Directive)

Warranty ..... 1 year parts and labor on defects in materials or workmanship

(1) Meets requirements of IEC-60068-2-1 and IEC-60068-2-2. Operating temperature may be extended to +55 °C with appropriate user-provided cooling solution. Contact SignalCore for recommended minimum airflow rates.
(2) Meets requirements of IEC-60068-2-1 and IEC-60068-2-2.
(3) Meets requirements of IEC-60068-2-56 and MIL-PRF-28800F, Class 3.
(4) Meets requirements of IEC-60068-2-56 and MIL-PRF-28800F, Class 3.
(5) Meets requirements of IEC-60068-2-27 and MIL-PRF-28800F, Class 3.
(6) Meets requirements of IEC-60068-2-27 and MIL-PRF-28800F, Class 3.
(7) Meets requirements of IEC-60068-2-64 and MIL-PRF-28800F, Class 3.
(8) Meets requirements of IEC-60068-2-64 and MIL-PRF-28800F, Class 3.

SignalCore, Inc,
13401 Pond Springs Rd.
Suite 100
Austin, TX 78729, USA
Phone: 512-501-6000
Fax: 512-501-6001